Aalto University
School of Science
Degree Program of Engineering Physics and Mathematics

Search space traversal using metaheuristics

Bachelor's Thesis

August 28, 2012

Mika Juuti

Aalto-yliopisto
Perustieteiden korkeakoulu
Teknillisen fysiikan ja matematiikan koulutusohjelma

KANDIDAATINTYÖN
TIIVISTELMÄ

Tekijä:	Mika Juuti
Työn nimi:	Search space traversal using metaheuristics
Päiväys:	28. elokuuta 2012
Sivumäärä:	17 + 1
Pääaine:	Systeemitieteet
Koodi:	F200
Vastuuopettaja:	Prof. Harri Ehtamo
Työn ohjaaja(t):	DI Ville Mattila

Työssä tarkastellaan kahta metaheuristiikkaa, Ant Colony Systemiä ja simuloitua jäähdytystä. Työn pääpaino on Ant Colony Systemin testaamisessa. Työssä esitellään metaheuristiikkojen yleiset toimintaperiaatteet sekä käyttöönotto kombinatorisissa tehtävissä. Ant Colony Systemiä ja simuloitua jäähdytystä testataan moniuloitteisella selkäreppuongelmalla. Ant Colony Systemiä testataan erilaisilla muurahaisten päätöksiin vaikuttavilla heuristiikoilla ja parametrivalinnoilla.

Testeissä algoritmit käyttäytyivät samankaltaisesti; suorituskyky heikkeni kun rajoitteita lisättiin, ja vastaavasti parani, kun esimerkiksi laskenta-aikaa lisättiin. Ant Colony System suoriutui testeissä paremmin kuin simuloitu jäähdytys. Ant Colony Systemissä sisäisen heuristiikan valinta vaikuttaa paljon tuloksiin. Myös parametrien valinnalla on merkitystä. Testeissä Ant Colony Systemin laskenta-ajan lisäämisellä ei ollut yhtä paljon vaikutusta kuin algoritmien sisäisten heuristiikkojen muuttamisella.

Avainsanat:	Metaheuristiikka, Ant Colony Optimization, Ant Colony System, Kombinatorinen Optimointi, Selkäreppuongelma, Simuloitu Jäähdytys
Kieli:	Englanti

Aalto-universitetet
Högskolan för teknikvetenskaper
Examensprogrammet för teknisk fysik och matematik

SAMMANDRAG AV
KANDIDATARBETET

Utfört av:	Mika Juuti
Arbetets namn:	Search space traversal using metaheuristics
Datum:	Den 28 augusti 2012
Sidoantal:	17 + 1
Huvudämne:	Systems Science
Kod:	F200
Övervakare:	Prof. Harri Ehtamo
Handledare:	DI Ville Mattila (Institutionen för matematik och systemanalys)

I det här kandidatarbetet undersöks två metaheuristiker: Ant Colony System och simulerad avkylning. Kandidatarbetet grundar sig huvudsakligen på testning av Ant Colony System. Metaheuristikers allmänna funktionsprinciper och tillämpning på kombinatoriska optimeringsproblem presenteras. I kandidatarbetet testas Ant Colony System och simulerad avkylning på det flerdimensionsionella kappsäcksproblemet. Ant Colony System testas med olika heuristiker och parameterinställningar, vilka vardera inverkar på myrornas beslutsprocess.

I kandidatarbetet visade det sig att algoritmerna beter sig på ett likartat sätt; algoritmernas prestanda minskade då antalet begränsande ekvationer ökade. På motsvarande sätt ökade prestandan då till exempel beräkningstiden ökades. I testerna presterade Ant Colony System bättre än simulerad avkylning. I Ant Colony System inverkade parameterinställningarna och speciellt valet av den interna heuristiken på testresultaten. Beräkningstiden hade mindre inverkan än den inre heuristiken i avseende på prestandan.

Nyckelord:	Metaheuristik, Ant Colony Optimization, Ant Colony System, Kombinatorisk Optimering, Kappsäcksproblemet, Simulerad Avkylning
Språk:	Engelska

Aalto University
School of Science
Degree Program of Engineering Physics and Mathematics

ABSTRACT OF
BACHELOR'S THESIS

Author:	Mika Juuti
Title of thesis:	Search space traversal using metaheuristics
Date:	August 28, 2012
Pages:	17 + 1
Major:	Systems Science
Code:	F200
Supervisor:	Professor Harri Ehtamo
Instructor:	Ville Mattila, M. Sc. (Tech.)

The thesis investigates metaheuristics, Ant Colony System and Simulated Annealing, with primary focus on Ant Colony System. The general principles of function and application to combinatorial optimization problems are presented. Ant Colont System and Simulated Annealing are tested on the multidimensional knapsack problem. Different internal heuristic and parameter choices are tested on Ant Colony System. Both change the preference of the decision agents, the ants.

The algorithms performed similarly in the tests. The performance of both algorithms degraded when the amount of constraints increased and improved when e.g. the computation time increased. Ant Colony System performed better than Simulated Annealing in the tests. The choice of the internal heuristic greatly affects the results in Ant Colony System. The choice of internal parameters are of significance as well. Increasing the computation time did not have as great an effect on the performance on the algorithms as changing the internal heuristic had.

Keywords:	Metaheuristics, Ant Colony Optimization, Ant Colony System, Combinatorial Optimization, Multiple Knapsack Problem, Simulated Annealing
Language:	English

Contents

Abbreviations and Acronyms

CO	Combinatorial Optimization
COP	Combinatorial Optimization Problem
TSP	Travelling Salesman Problem
SA	Simulated Annealing

Ant Colony System (ACS)

ACO	Ant Colony Optimization
ACS	Ant Colony System
$\tau_{i,j}$	Pheromone trail from node i to j
τ_0	Initial (and also minimum) level of pheromone on any trail

1 Introduction

Metaheuristics are a group of approximate algorithms that use some heuristic method in a higher level framework that aims at effectively and efficiently exploring a search space. The word metaheuristic comes from two Greek words. The word *heuristic* comes from the word *heuriskein* that means "to find", while *meta* is Greek for "beyond, in an upper level", Blum and Roli (2003). Osman and Laporte (1996) define metaheuristics as:

> They[Metaheuristics] are designed to attack complex optimization problems where classical heuristics and optimization methods have failed to be effective and efficient. A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions.

Exploration is the act of expanding the searched solution space, while *exploitation* is the act of using accumulated knowledge to select areas to explore next. Sometimes *diversification* and *intensification* are used in stead; often with a bigger emphasis on medium to long term strategies, Blum and Roli (2003). Metaheuristics are not problem specific. http://www.metaheuristics.net (2012) declares that

> A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make them adapted to a specific problem.

Combinatorial optimization is a term denoting optimization problems where clear discrete or combinatorial structures arise, Kreyszig (1999). Beasley (2012) claims that integer programming nowadays is called combinatorial optimization, indicating that the number of possible solutions increases very much (a combinatorial increase) as the problem size increases. The objective for any optimization problem is to find the best solution for a problem – the (globally) optimal solution. In combinatorial optimization problems, the solution space is either a finite or a countably infinite discrete set.

Metaheuristics are used in combinatorial optimization problems to quickly produce good-quality solutions. Many COPs are still not solved in reasonable time and require good metaheuristic methods, Fidanova (2006). Very hard combinatorial optimization problems belong to the time complexity class **NP**-hard, Blum and Roli (2003). These problems

are not conceptually hard; the optimal solution can be found by enumeration. The rapid increase in possible solutions when the problem size increases makes these problems computationally hard.

NP-hard problems are at least as difficult to solve as **NP**-complete problems. **NP**-completeness in turn means that it can be calculated by a *non*deterministic Turing machine in polynomial time. This means that such a problem could be calculated in with a polynomial time bound if the program doing the calculations happens to "guess" correctly, Papadimitriou (1994). Nondeterministic Turing machines do not however describe a realistic computation model. Instead, most of the time a **NP**-complete problem will be solved in a time that does not appear to be polynomially bounded.

This thesis investigates an ant colony optimization algorithm and a simulated annealing algorithm using the zero-one multidimensional knapsack problem (MKP), which is a **NP**-hard problem. This thesis stresses algorithm presentation. Tests on the performance of the algorithms are done at the end of the paper.

This thesis is limited to analysis of discrete solution space (search space) problems. Hybridization approaches to metaheuristics are not investigated.

2 Theory and background

In this section, the multidimensional knapsack problem is first presented in order to describe the metaheuristics. Then the ACO algorithm and the SA algorithm solving this problem are introduced.

2.1 Multidimensional knapsack problem

The multidimensional knapsack problem can be stated as

$$\max \sum_{j=1}^{n} p_j x_j \tag{1}$$

such that

$$\sum_{j=1}^{m} r_{ij} x_j \leq c_i, i = 1, .., m \tag{2}$$

$$x_j \in \{0, 1\}, j = 1, .., n \tag{3}$$

where the decision variable x_j represents whether the jth item is included in the knapsack.

Equation (1) is the objective function that represents how much the included items are of worth. Equations (2) are the m constraints in the problem. c_i is the capacity (budget) in dimension i and r_{ij} is the resource consumption of item j in dimension i. In order words, there are m constraints and n items. It is desired to maximize the sum of the *profits* p_j by including an optimal subset of the n items into the knapsack. Equation (2) states that the summed *weights* r_{ij} of the items should not exceed any of the constraints.

MKP is a generalization of the knapsack problem, where there are multiple constraints on each item that can be added to the knapsack. It is a resource allocation problem, where there is a set of items, each item with its unique resource consumption and profit. The objective is to maximize the sum of the profits of the included items without violating the resource constraints. MKP appears as a subproblem in many other COPs, including the vehicle routing problem and the generalized assignment problem, Fidanova (2006). MKP sees direct use in cargo loading problems, cutting stock problems, budget control, bin packing and financial management. MKP belongs to the complexity class **NP**-hard.

2.2 Ant Colony System

Ant Colony Optimization is an umbrella term for a range of metaheuristics using a population of simple agents (*ants*) and state transition rules governed partially by *a priori* heuristic information and partially by a learning component, the *pheromone trail*. The differences between the different Ant Colony Optimization metaheuristics are mainly found in how these pheromone trails are updated. Ant Colony System (ACS) was presented in Gambarella and Dorigo (1996) and has become one of the most widely used Ant Colony Optimization metaheuristics.

In order to use ACS to solve a COP it is necessary to describe the problem using a graph, since ACO algorithms solve problems by using artificial ants that construct solutions by traversing graph arcs. A graph may map any sort of relation between two or more objects. The objects are often referred to as *vertices* (or nodes) and the relations between the objects as *edges*. A graph is considered directed if every edge $e(i, j)$ contains a direction from its "initial point" i to its "terminal point" j, Kreyszig (1999). Directed edges are sometimes referred to as arcs. Figure (1) depicts a simple directed graph.

In order to map MKP to a graph it is necessary to decide which parts of MKP correspond to nodes and arcs. In this thesis every item is mapped as a node and arcs fully connect nodes. A fully connected arc means that after selecting an item i, the next selected node might be j if there are enough resources and item j has not yet been chosen, Fidanova (2006). In this case the edge $e(i, j)$ can be traversed. A path through the graph coincides then with a feasible solution to the COP.

Additionally, it is necessary to create a heuristic value function for the ants. This will

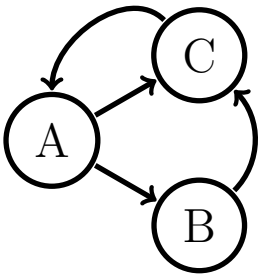

Figure 1: A directed graph with 3 vertices and 4 edges. The set of vertices $V = \{\text{A,B,C}\}$. The set of edges $E = \{\text{(A,B), (A,C), (B,C), (C,A)}\}$

help the ants navigate in the graph. In this manner, the heuristic may be thought of as being a map or compass for the ants. It is the heuristic's task to steer the ant in the approximate right direction. In MKP this should mean something like "it is good to include items that take little space and are of high value". In the heuristic value function, higher values are more desirable than lower values.

2.2.1 State transition

In ACS the rule according to which ants move is called the *pseudo-random proportional action choice rule* or the state transition rule. This rule has a certain fixed probability of choosing the next item greedily, i.e. the item that according to the available information seems to be the best item to add to the knapsack. If the greedy alternative is not chosen, the probability of choosing any other feasible item is proportional to the value of the arc from the previously added item to this item. The state transition rule for ant k proceeding from node r to node s can be stated as

$$s = \begin{cases} \arg \max\limits_{z \in J_k(r)} \{[\tau(r,z)][\eta(r,z)]^\beta\} & \text{if } q \leq q_0 \\ S & \text{otherwise} \end{cases} \tag{4}$$

where $J_k(r)$ is the set of allowed nodes from node r, η is the heuristic value function from the current node r to node z, $\tau(r,z)$ is the pheromone trail from node r to node z and β is a coefficient that weights the importance of heuristic value function. q is a random variable drawn from a uniform distribution $U(0,1)$. With a probability q_0 the best perceived value is chosen, otherwise the next node is chosen according to another decision rule S.

The pheromone trail can be viewed as the collective memory of all agents. In ACS every trail has *at least* a pheromone trail level of τ_0. Higher pheromone trail levels mean that

at least at some point during the course of the algorithm, this particular arc has been a part of the best solution found so far.

The value $\tau(a,b)][\eta(a,b)]^\beta$ is will henceforth be called the *arc value* for arc (a,b) (or the value of the arc (a,b)) for the sake of simplicity. Here, $\eta(a,b)$ is the heuristic value function for the arc between node (item) a and node (item) b. As suggested by Fidanova (2006) $\eta(a,b)$ for the MKP might be

$$\eta_1(a,b) = p_b\big(\sum_{i=1}^{m} \frac{r_{ib}}{c_i}\big)^{-1} \tag{5}$$

According to the Equation (5), the value is higher the higher the profit of including the item is and lower the more space it requires. This can be seen since $\frac{r_{ij}}{c_i}$ is the ratio of the available space that the item claims in dimension i. For example, if i = 2, j = 3, $r_{ij} = 2$, $c_2 = 5$, then the ratio $\frac{r_{ij}}{c_i}$ can be interpreted as a statement that the proposed item 3 would claim 40% of the total available resource in period 2.

This heuristic is directly proportional to the profit of the item and inversely proportional to the sum of the proportionalized weights of each restriction. This way, good items are added to the knapsack, as discussed earlier. Figure (2) illustrates the calculation of a single ant's move from one node to another in the beginning of an algorithm run.

Another heuristic that would add items in the order of how much additional value they contribute is tested too. This heuristic is

$$\eta_2(a,b) = p_b \tag{6}$$

In Equation (4), if the random variable $q > q_0$ the greedy alternative is not chosen and instead the probability of choosing a particular arc (a,b) is proportional to the arc value divided by the sum of all the values of the feasible arcs in the neighbourhood of node a. The probability of ant k choosing arc (a,s) is positive if the arc (a,s) is a feasible arc; otherwise it is equal to zero:

$$p_k(r,s) = \begin{cases} \dfrac{[\tau(r,s)][\eta(r,s)]^\beta}{\sum\limits_{z \in J_k(r)} [\tau(r,z)][\eta(r,z)]^\beta} & \text{if s} \in J_k(r) \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

here J_k is the set of feasible destinations for ant k; items that will not make the partial solution violate any constraints should they be included in the partial solution created by ant k.

The pseudo-proportional action choice rule promotes solutions in the proximity of the

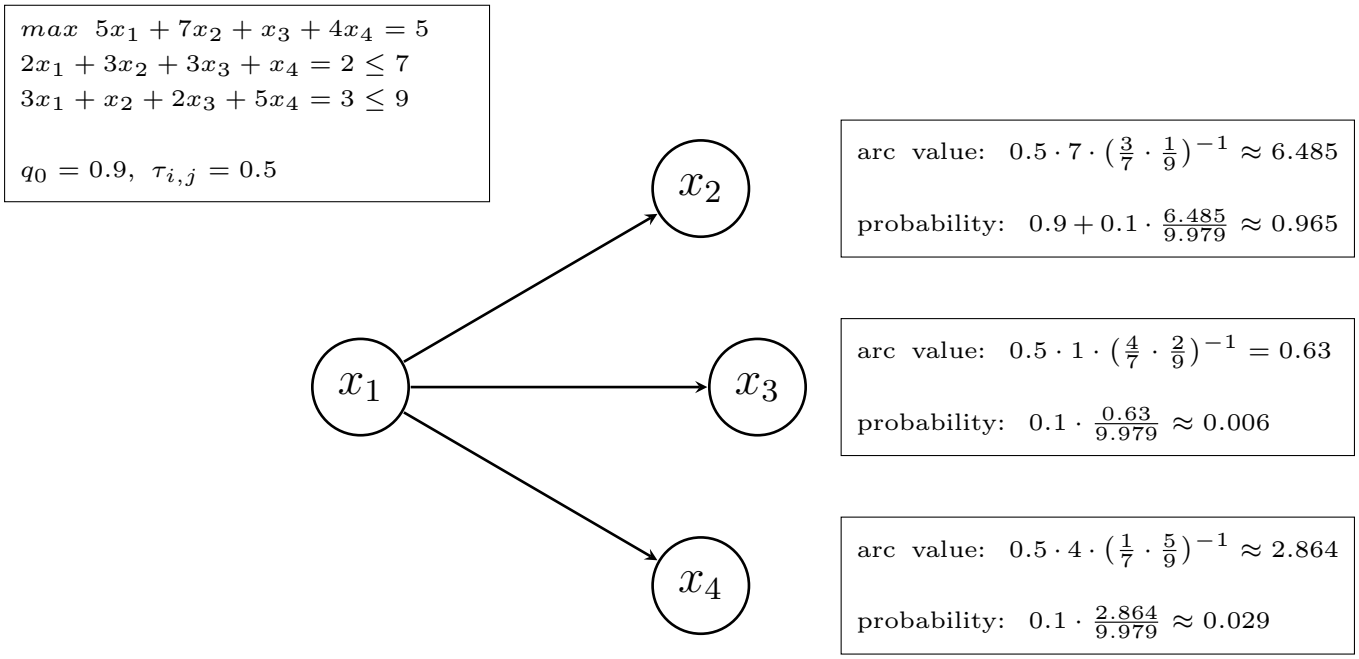

Figure 2: State transition from node x_1. The problem has two constraints. The arc values are calculated from x_1 using Equation (4) with $\beta = 1$. The actual probabilities for the arcs to be chosen are calculated with Equations (4) and (7). Arc (x_1,x_2) has the highest arc value and will be automatically chosen with a probability of $q_0 = 0.9$. Otherwise, the probability of choosing a node is proportional to the arc value divided by the sum of the arc values.

greedy alternative. The ant relies on the collective memory and chooses the arc with the maximum arc value with probability slightly higher than q_0.

If there are no elevated pheromone trail levels in the neighbourhood of the ant, its choice will mainly be influenced by the greedy alternative, having a probability of q_0 or higher of being chosen. This is the case in the beginning of any ACS algorithm and in cases where the ant has ended up in a rural location in the state space either through initiation or by choosing the location through the pseudo-proportional action choice rule.

2.2.2 Iteration

ACS algorithms can coarsely be said to consist of two loops and a pheromone update is applied in the last stage of both iterations. The outer loop runs until a stopping criterion is encountered. The inner loop iterates a solution construction procedure for every ant in the population. Stopping criterions for the outer loop might be a set number of iterations, closeness to the optimal solution (if information is available) or exceeding a time limit. In the inner loop solutions are constructed for each ant, one ant at a time, using the state transition rule presented in the previous chapter. Each time after an ant has constructed its complete solution, the pheromone trail of the solution (i.e. for every arc in the solution) is decreased by formula:

$$\tau_{r,s} = (1 - \rho)\tau_{r,s} + \rho\tau_0 \tag{8}$$

where ρ is between 0 and 1. Equation (8) says that the value of pheromone trail is updated to become a value between its old value and the initial pheromone level. ρ determines how much of the trail is dissipated; $\rho = 1$ means that the trail forgets all its pheromone and is reinitialized to its initial value τ_0. The trail dissipation is part of the metaheuristics exploration policy. Decreasing this value means that this trail has a lower chance of being selected in the future. Figure (3) illustrates the local pheromone update procedure with an example.

Once all ants have constructed their solutions, the pheromone trail of the best solution is reinforced. The reinforced trail is chosen either from the current iteration or from entire run of the algorithm. Since MKP is a maximization problem this rule can be written as

$$\tau_{r,s} = (1 - \alpha)\tau_{r,s} + \alpha F_{best} \tag{9}$$

where F_{best} is this best solution found. If the problem is a minimization problem, F_{best} can be replaced by $1/F_{best}$, Fidanova (2006). Reinforcement of the best found solution has been used in the ACS algorithm tested in this paper. Figure (4) exemplifies the global pheromone update procedure.

The pseudocode of ant colony system for the multidimensional knapsack problem is in Algorithm 1. Montemanni et al. (2004) provide a similar pseudocode for the vehicle routing problem.

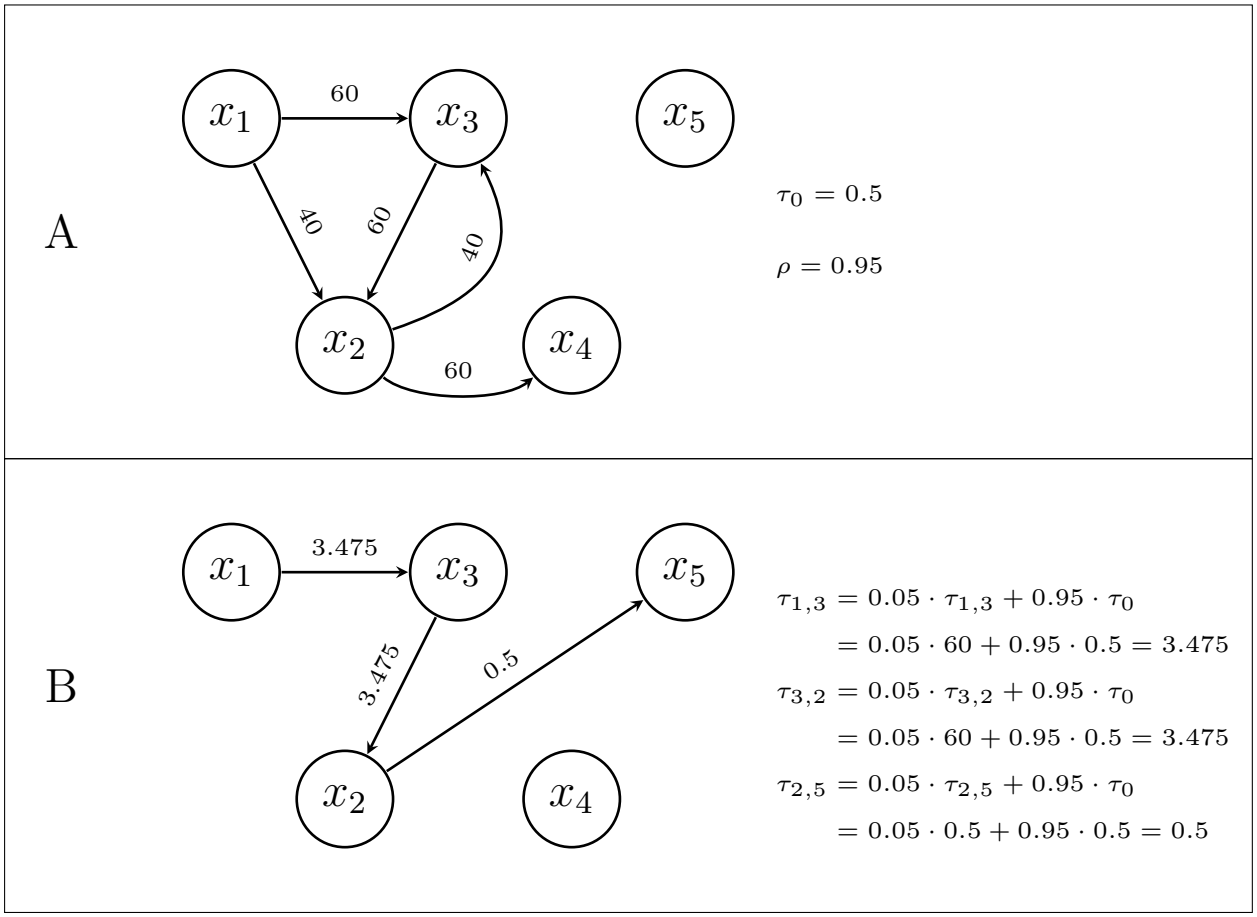

Figure 3: An illustration of the local pheromone updating rule. Subfigure A depicts the pheromone trails that are not equal to their initial value τ_0. In subfigure B, an ant has constructed a solution (x_1,x_3,x_2,x_5). The pheromone trails are update according to Equation (8) with $\rho = 0.95$. The pheromone trail on arc (x_2,x_5) remains unchanged since it did not have an elevated pheromone trail level.

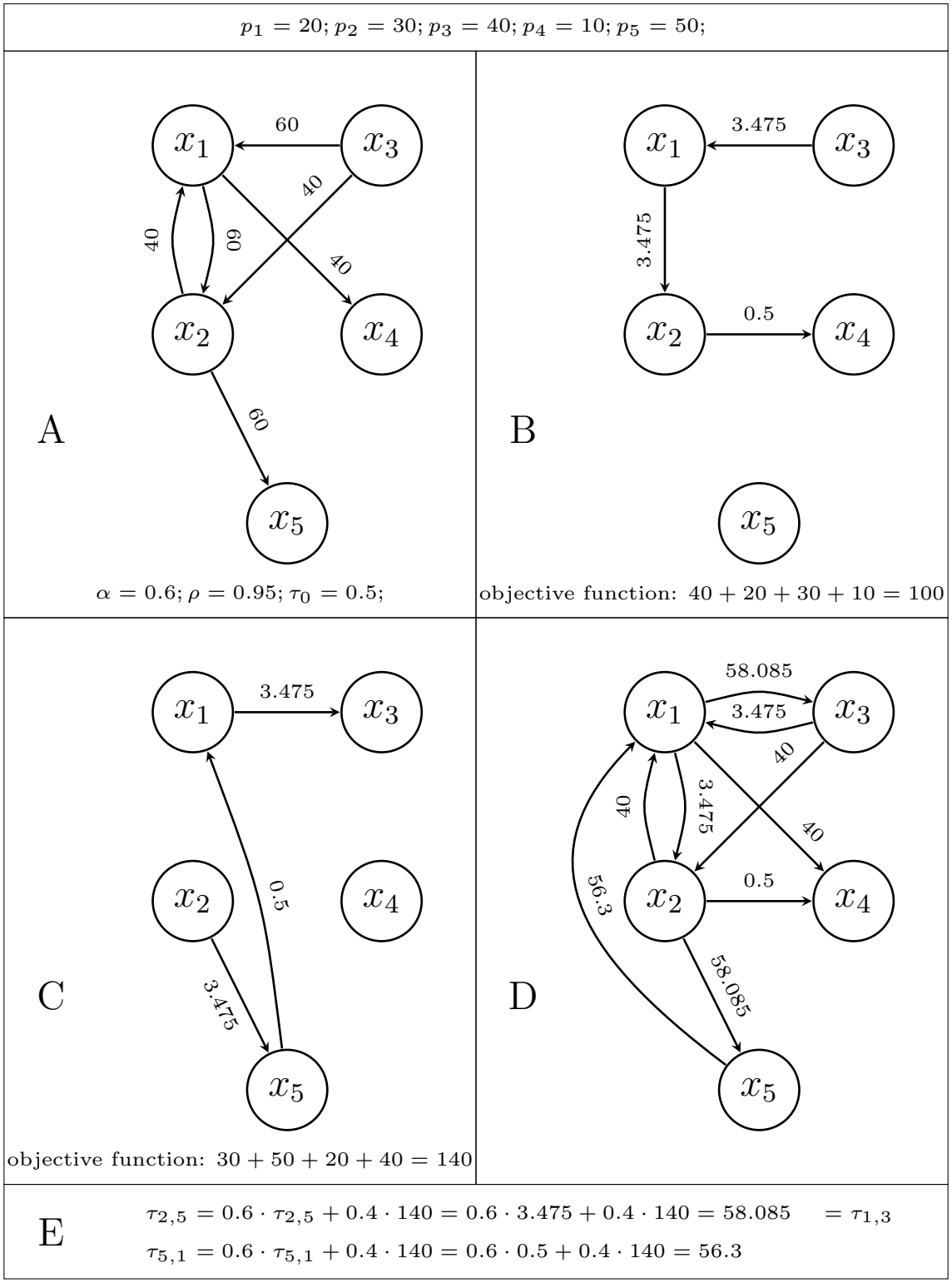

Figure 4: An example of the global pheromone update rule. Subfigure A depicts the pheromone trails between the five nodes that are not equal to the initial value $\tau_0 = 0.5$ in the beginning. In subfigure B, an ant has constructed a solution (x_3,x_1,x_2,x_4) of the value 100 and the pheromone trails are updated locally. In subfigure C, another ant constructs a solution (x_2,x_5,x_1,x_3) of the value 140 and pheromones are updated locally again. In subfigure D, the best pheromone trail (in subfigure C) is reinforced according to Equation (9) with $\alpha = 0.4$. These global pheromone trail update calculations are presented at E.

```
BestCost := ∞;
for each arc (i,j) do
    τ_{i,j} = τ_0;
end
while computation time < computation limit do
    for k := 1 to m do
        while Ant k has not completed its solution do
            Select new item j; Update the trail level τ_{i,j} (Equation (8));
        end
        Cost := Cost of current solution;
        if Cost < BestCost then
            BestCost = Cost;
            BestSol = current solution;
        end
    end
    for each move (i,j) in solution BestSol do
        Update the trail level τ_{i,j} (Equation (9));
    end
end
return BestSol,BestCost
```

Algorithm 1: Pseudocode for the ant colony system algorithm.

2.3 Simulated Annealing

Simulated Annealing (SA) is memoryless metaheuristic that continuously improves one solution in generations. The next generation is accepted if it gives a better solution than the previous accepted solution. Solutions that decrease the value of the objective function are accepted with a certain probability. This probability is dependant on a *temperature* control parameter and the probability usually decreases towards the end of the algorithm. Selecting these moves allows SA to escape local minima. The probability of selecting a move that decreases the value of the objective function from solution s to solution s' is commonly defined as a Boltzmann distribution

$$p = e^{(f(s')-f(s))/t} \tag{10}$$

where f is the objective function and t is the temperature parameter. Here t should be of the order of the maximum difference between neighbouring solutions, Qian and Ding (2007).

A flowchart describing the simulated annealing algorithm is presented in Appendix A. In this algorithm, the temperature is decreased at each iteration by multiplication with the temperature control parameter α. α should therefore be between 0 and 1. In the tests $\alpha = 0.85$. The execution continues until a low enough temperature ϵ is reached. $\epsilon = 10^{-5}$ in the tests. In SA, M is the Markov-chain length that describes how many state changes are made at each temperature, Qian and Ding (2007). During the state change a new random item is included in the knapsack.

In case the solution became infeasible, one of the included items will be randomly dropped from the knapsack. This continues until a feasible solution has been found. Once a feasible solution has been found (either by dropping or including an item any way), the current solution's objective function value is compared against the *accepted* solution's objective function value. If the current solution is better than the accepted solution it will become the new accepted solution. Otherwise it will be accepted with the probability in Equation (10).

The temperature control loop is restarted by an outer loop, until enough time has passed since the beginning of the execution. In the tests this time has been between one second and 5 minutes. Iteration continues from the previous accepted solution upon restart. In this manner, the cooldown schedule of the SA algorithm in this paper can be thought of as dynamic.

3 Performance Tests

ACS and SA algorithms were tested by comparing their performance in MKP when run on different execution times. Performance dependence on the number of selectable items n, the number of constraints m and on the "easiness" of the problem; the tightness ratio α were tested. The tightness ratio is determined by

$$\alpha = \frac{c_i}{\sum_{j=1}^{n} r_{ij}}, \forall i \tag{11}$$

where c_i is the resource constraint in dimension i and r_{ij} is the resource consumption of item j in dimension i. The tightness ratio was equal for every dimension i in the test problems. The tightness ratio can be though of as the coefficient that the sum of the item weights must be multiplied by to be the constraint limit in any dimension. In the test cases the tightness ratio is equal in all m dimensions. A tightness ratio of 1 means that all items will fit the knapsack and a tightness ratio of 0 means that no items will fit into the knapsack. Neither of these are interesting scenarios.

All the test instances were fetched from the OR library, Chu and Beasley (1998). Each test scenario is measured in groups of five similar problems that have the same n, m, run time and α, except for the aspect to be tested. The performance is normalized against either the optimal Integer Programming solution or the relaxed Linear Programming solution. The overall performance of the algorithm is given as a percentage. It is calculated by running each test case five times and calculating the average of each test case's best performance.

It is also tested if setting the pheromone trail $\tau_{b,a}$ to the same level as $\tau_{a,b}$ for any pair of items a and b during all pheromone changes, has an affect on the results. MKP is an unordered problem, meaning that it doesn't affect the goodness of a solution if an item has been added at a later point than another; they are both in the knapsack. Thus it might be in the interest of better solutions to have a symmetric pheromone update policy.

All tests were concluded on a Pentium T4400 2.2GHz computer running MATLAB 2009 on Windows 7. The computer has 4GB RAM.

3.1 Time Dependency

Time dependency of the ACS and simulated annealing algorithms were examined on 5 problems, each with 250 items, 10 constraints and a tightness ratio of 0.50. The tests were performed with 1, 10, 30, 120 and 300 second execution times and the best results from the five runs were included in the statistics. As for ACS, different parameters, pheromone update rules and heuristics were tested. The goal was to find any particular

patterns where the algorithms performed exceptionally well, or very poorly.

Table 1: The average of the best performances at different execution times compared to the integer programming optimal solution.

Row	Algorithm	1 s	10 s	30 s	120 s	300 s
1	Ant Colony System, symmetric, $\rho = 0.95$, $q_0 = 0.9$, η_1	0.9669	0.9710	0.9766	0.9769	0.9809
2	Ant Colony System, asymmetric, $\rho = 0.95$, $q_0 = 0.9$, η_1	0.9663	0.9727	0.9728	0.9788	0.9788
3	Ant Colony System, symmetric, $\rho = 0.9$, $q_0 = 0.8$, η_1	0.9676	0.9672	0.9725	0.9739	0.9756
4	Ant Colony System, asymmetric, $\rho = 0.9$, $q_0 = 0.8$, η_1	0.9626	0.9705	0.9750	0.9760	0.9763
5	Ant Colony System, symmetric, $\rho = 0.5$, $q_0 = 0.1$, η_2	0.8803	0.8708	0.8961	0.8939	0.8944
6	Ant Colony System, asymmetric, $\rho = 0.5$, $q_0 = 0.1$, η_2	0.8785	0.8710	0.8931	0.8957	0.8990
7	Simulated Annealing, $M = n$	0.8791	0.8822	0.8848	0.8903	0.8935

Comparing the performance of row 1 and 2, row 3 and 4, row 5 and 6 in Table 1 with each other, it appears that whether the pheromone update rule is applied symmetrically or not does not affect the result very much. Both symmetric and asymmetric versions achieve results of the same order. It seems as if the symmetric variant attains slightly better solutions at a one second execution time. This advantage largely seems to disappear at ten seconds. The effect of the different heuristic and parameter changes on the other hand significantly impact the overall closeness to the integer programming global minimum. It appears as if η_2-algorithms (rows 5-6) stagnate around 0.90, however η_1-algorithms reach results as high as 0.98 (rows 1-4). In comparison, the results for the SA algorithm on row 7 are in the range 0.88-0.89.

A higher pheromone trail evaporation (ρ) coupled with a higher susceptibility of choosing the greedy alternative (q_0) on rows 1 and 2 outperform the lower counterparts on rows 3 and 4. Higher values of q_0 and ρ contribute to a higher exploration effect, since pheromone trail information about the greedy alternative is (with a high probability) quickly used away. This would cause a smaller stagnation effect in the results.

Overall, it seems as if the performance of the algorithms are not particularly sensitive to increases in computation time in the time scale between 1 and 300 seconds. The largest performance improvements are found in row 6 with ACS algorithm with the heuristic value

function η_2, improving its result from 10 s to 300 s by 2.8 percentage points. The random factor is quite visible in rows 5 and 6 as the ACS algorithms degrade in performance from their 1 second run to their 10 second run.

3.2 Dependency on items

These tests were concluded with a 120 s execution time, 10 constraints and a tightness ratio of 0.5, with problems with either 100, 250 or 500 items.

Table 2: The average of the best performances with different numbers of items compared to the linear programming relaxed optimal solution.

Algorithm	$n = 100$	$n = 250$	$n = 500$
Ant Colony System, symmetric, $\rho = 0.95$, $q_0 = 0.9$, η_1	0.9807	0.9741	0.9817
Ant Colony System, symmetric, $\rho = 0.9$, $q_0 = 0.8$, η_1	0.9765	0.9721	0.9775
Ant Colony System, symmetric, $\rho = 0.5$, $q_0 = 0.1$, η_2	0.9081	0.8869	0.8821
Ant Colony System, asymmetric, $\rho = 0.5$, $q_0 = 0.1$, η_2	0.9099	0.8887	0.8813
Simulated Annealing, $M = n$	0.9058	0.8864	0.8739

Looking at Table 2 it appears as if ACS heuristic η_1 does not degrade in performance when the number of items increases. Heuristic η_2-algorithms on rows 3 and 4 and the SA algorithm see slight improvements in performance as the number of items decrease. The performance of the heuristic η_2-algorithms are quite similar to the simulated annealing algorithm's result. Higher ρ and q_0 values give better results even in this test. Overall the number of items does not affect the performance of any algorithm very strongly.

3.3 Dependency on constraints

Tests on performance changes with different number of constraints are carried out with a 120 s execution time, 250 items, and a tightness ratio of 0.50.

Looking at Table 3, it is obvious that all algorithms perform worse as the number of constraints increases. ACS η_1-algorithms and the SA are affected more strongly than the ACS η_2 algorithms. The symmetric ACS η_2-algorithm had the smallest degradation and improvement in performance. The ACS η_1-algorithm on row 1 came very close to the

Table 3: The average of the best performances with different number of constraints compared to the linear programming relaxed optimal solution.

Algorithm	$m = 5$	$m = 10$	$m = 30$
Ant Colony System, symmetric, $\rho = 0.95$, $q_0 = 0.9$, η_1	0.9895	0.9741	0.9669
Ant Colony System, symmetric, $\rho = 0.9$, $q_0 = 0.8$, η_1	0.9822	0.9721	0.9629
Ant Colony System, symmetric, $\rho = 0.5$, $q_0 = 0.1$, η_2	0.8880	0.8869	0.8837
Ant Colony System, asymmetric, $\rho = 0.5$, $q_0 = 0.1$, η_2	0.8931	0.8887	0.8838
Simulated Annealing, $M = n$	0.8901	0.8864	0.8773

linear programming relaxed optimal solution. Higher ρ and q_0 values still outperform lower ones, as can be seen when comparing rows 1 and 2.

3.4 Tightness Ratio

Tests on how the tightness ratio affects performance were conducted by letting the number of items stay at 250 and keeping the number of constraints at 10. The execution times of the algorithms were 2 minutes.

Table 4: The average of the best performances at different levels of tightness ratio compared to the linear programming relaxed optimal solution.

Algorithm	$\alpha = 0.25$	$\alpha = 0.50$	$\alpha = 0.75$
Ant Colony System, symmetric, $\rho = 0.95$, $q_0 = 0.9$, η_1	0.9643	0.9741	0.9896
Ant Colony System, symmetric, $\rho = 0.9$, $q_0 = 0.8$, η_1	0.9564	0.9721	0.9870
Ant Colony System, symmetric, $\rho = 0.5$, $q_0 = 0.1$, η_2	0.8365	0.8869	0.9469
Ant Colony System, asymmetric, $\rho = 0.5$, $q_0 = 0.1$, η_2	0.8318	0.8887	0.9460
Simulated Annealing, $M = n$	0.8348	0.8864	0.9005

Looking at Table 4, it seems clear that the tightness ratio affects the algorithms quite strongly. A lower tightness ratio means that fewer items fit into the knapsack, as there are tighter resource constraints. Heuristic η_1 (ACS algorithms on row 1 and 2) does not degrade in performance as heavily as others when α decreases. These algorithms also have improved performance as α approaches 1.

ACS heuristic η_2 is the most affected by α, greatly improving its performance as α increases. This might be accredited to ACO algorithms being constructive algorithms in the manner that they continue building upon a solution until no more items fit into the their partial solution. And then they start again from the bottom. It seems natural that this sort of addition of items would enable good results when a majority of the items fit into the knapsack.

Simulated annealing does not perform as well as the ACS algorithms when $\alpha = 0.75$. This could be accredited to SA being a trajectory method; improving a single existing solution until the very end of the algorithm.

4 Conclusions

In this thesis, a combinatorial optimization problem has been solved with two different metaheuristics. ACS manages many simultaneous solutions that communicate indirectly, while SA continuously improves its solution by manipulating a parameter called the temperature. The manner in which these metaheuristics attack COPs varies quite much.

Application of an ACO algorithm on a COP requires mapping the problem into an appropriate graph that allows the ants to build solutions by crossing arcs. A good heuristic value function for the ants to exploit is also necessary. Application of simulated annealing on the other hand requires a manner in which to manipulate a vector of binary values in a manner appropriate to the COP.

During the tests ACS η_1-algorithms dominated all other algorithms. Higher values in the pheromone trail evaporation ρ and in the probability of choosing the greedy alternative q_0 gave even better results.

Finally the results for the ant colony system algorithm are heavily dependant on the heuristic value function; it is important that a good heuristic value function is selected, and that this behaves well with the parameter values in the algorithm (ρ,q_0,...). No algorithm was particularly dependant on how long the execution time was, so a parameter optimization could be performed with a low execution time. Later on, execution time can be increased if slightly better results are desired at the end of the run of the algorithm.

References

J.E. Beasley. Or-notes, 2012. URL `http://people.brunel.ac.uk/~mastjjb/jeb/or/ip.html`.

Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, pages 268–308, 2003. ISSN 0360-0300.

P.C. Chu and J.E. Beasley. Multidimensional knapsack problem, 1998. URL `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html`.

Stefka Fidanova. *Ant Colony Optimization and Multiple Knapsack Problem.* Handbook of Research on Nature-inspired Computing for Economics and Management, Hershey, PA, 2006. ISBN 1591409845.

Luca Gambarella, Maria and Marco Dorigo. Solving symmetric and asymmetric tsps by ant colonies. *IEEE Conference on Evolutionary Computation*, pages 622–627, 1996.

http://www.metaheuristics.net, 2012. URL `http://www.metaheuristics.net/`.

E. Kreyszig. *Advanced Engineering Mathematics 8th Edition.* John Wiley & Sons, 1999.

Roberto Montemanni, Luca Gambardella, Maria, Andre-Emilio Rizzoli and Alberto V. Donati. Ant colony system for a dynamic vehicle routing problem. *Journal of Combinatorial Optimization*, 10:327–343, 2004. ISSN 1382-6905. Issue 4.

I. H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63:513–613, 1996.

Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley Publishing Company, Inc., 1994. ISBN 0-201-43082-1.

Fubin Qian and Rui Ding. Simulated annealing for the 0/1 multidimensional knapsack problem. *Numerical Mathematics - English Series* -, Vol 16:320–327, 2007. ISSN 1004-8979.

A Simulated Annealing flowchart