# Metaheuristics and Combinatorial Optimization

by

Bruce L. Golden
Robert H. Smith School of Business
University of Maryland

---

Graph Theory, Algorithms, and Applications
Erice, Italy, September 11, 2008

# Outline of Lecture

- A brief review of metaheuristics and their contributions to combinatorial optimization
  - Far from comprehensive
  - Biased in the direction of my interests
- A discussion of the key questions posed by research in the field of metaheuristics and partial answers
- A review of some of my recent work in the area
- A short list of new questions for researchers to consider

# The Elegance of Simulated Annealing

- Simulated annealing (SA) emerges from original paper by Kirkpatrick, Gelatt, and Vecchi (1983) in Science
- It has had a major impact on the field of heuristic search
  - simple
  - robust
  - clever analogy to statistical mechanics
- Two intuitive parameters
  - starting temperature
  - reduction rate or number of repetitions

# Simulated Annealing Algorithm

Step 1. Compute an initial feasible configuration T and choose an initial temperature $\tau$ and a repetition factor $r$

Step 2. As long as the stopping criterion is not satisfied, perform the sub-steps

a. Do the following $r$ times

(i) Perform a random modification of the current configuration to obtain the configuration T' and let $\Delta = c(T') - c(T)$, the difference in configuration lengths

(ii) If $\Delta < 0$ (improvement), then set T = T'. If not, set T = T' with probability = $e^{-\Delta/\tau}$

b. Update $\tau$ and $r$

Step 3. Output the final (or best) solution

# Significance of Simulated Annealing

- Construction heuristics, improvement heuristics, and random restart heuristics had been around before 1983

- But, simulated annealing addressed the question

> Why should we only consider downhill moves?

# Alternatives to Simulated Annealing

- Deterministic Variants
  - Threshold accepting – Dueck and Scheur (1990)
  - Record-to-record travel – Dueck (1993)
  - Great deluge algorithm – Dueck (1993)
- These are comparable to SA in performance and address the question

> Can we mimic SA without generating random numbers?

# Similar to Simulated Annealing

- Demon Algorithms
  - Introduced by Wood and Downs (1998)
  - Extensions by
    - Pepper, Golden, and Wasil (2002)
    - Chandran, Golden, and Wasil (2003)
  - Some variants are deterministic
  - Others are stochastic
  - The best variants outperform SA

# The Arrival of Tabu Search

- Glover's classic paper "Future Paths for Integer Programming and Links to Artificial Intelligence" appears in 1986 in C & OR
- He discusses controlled randomization as an old strategy for overcoming local optimality
- He argues that SA is an example of controlled randomization
- He suggests tabu search (TS) as a new strategy and an alternative to SA

# Metaheuristics and Tabu Search

- Glover fleshes out the details of TS and its advantages with respect to SA
  - TS prevents cycling
  - TS prevents certain moves from being (easily) reversed
  - Whereas SA is a passive search, TS is a more active search
- TS addresses the question

> Why leave things to chance?

# The Evolution of Genetic Algorithms

- Early work on adaptive systems and genetic algorithms (GAs) began in the 1960s
- Holland's influential book Adaptation in Natural and Artificial Systems is published in 1975
- In the 1980s, genetic algorithms (GAs) were first applied to combinatorial optimization problems
- These GAs used simple operators for crossover and mutation and were not competitive with state-of-the-art heuristics

# More on Genetic Algorithms

- Goldberg's book Genetic Algorithms is published in 1989
- In the 1990s, more sophisticated operators were proposed and hybridization (e.g., using local search as a mutation operator) became popular
- The effectiveness of GAs improved
- Bean (1994) proposed the widely applicable random keys representation of a feasible solution to a combinatorial optimization problem

# GAs are Appealing

- GAs have some attractive properties
  - The framework is simple and elegant
  - There is a beautiful analogy to natural selection and survival of the fittest
  - Nonlinearities are easily handled
  - GAs are robust
  - Parallel computing and GAs are a good match
- GAs address the question

> How can one build a metaheuristic based on the notion of competition?

# Tabu Search with Strategic Oscillation

- For an introduction to strategic oscillation, see Glover (1989)
- Kelly, Golden, and Assad (1993) applied this idea to large-scale controlled rounding
- Finding feasible solutions to controlled rounding problems is, in itself, NP-complete
- These problems are important to the U.S. Bureau of the Census

# More on Strategic Oscillation

- Strategic oscillation allows us to consider (and penalize) intermediate solutions that are infeasible
- We were able to obtain excellent solutions to three-dimensional controlled rounding problems
- Strategic oscillation addresses the question

Why not consider infeasible intermediate solutions?

# Fine-tuned Learning Metaheuristic

0. Start with original inputs

1. Average, aggregate, approximate, or smooth the inputs

2. Perform PROCEDURE

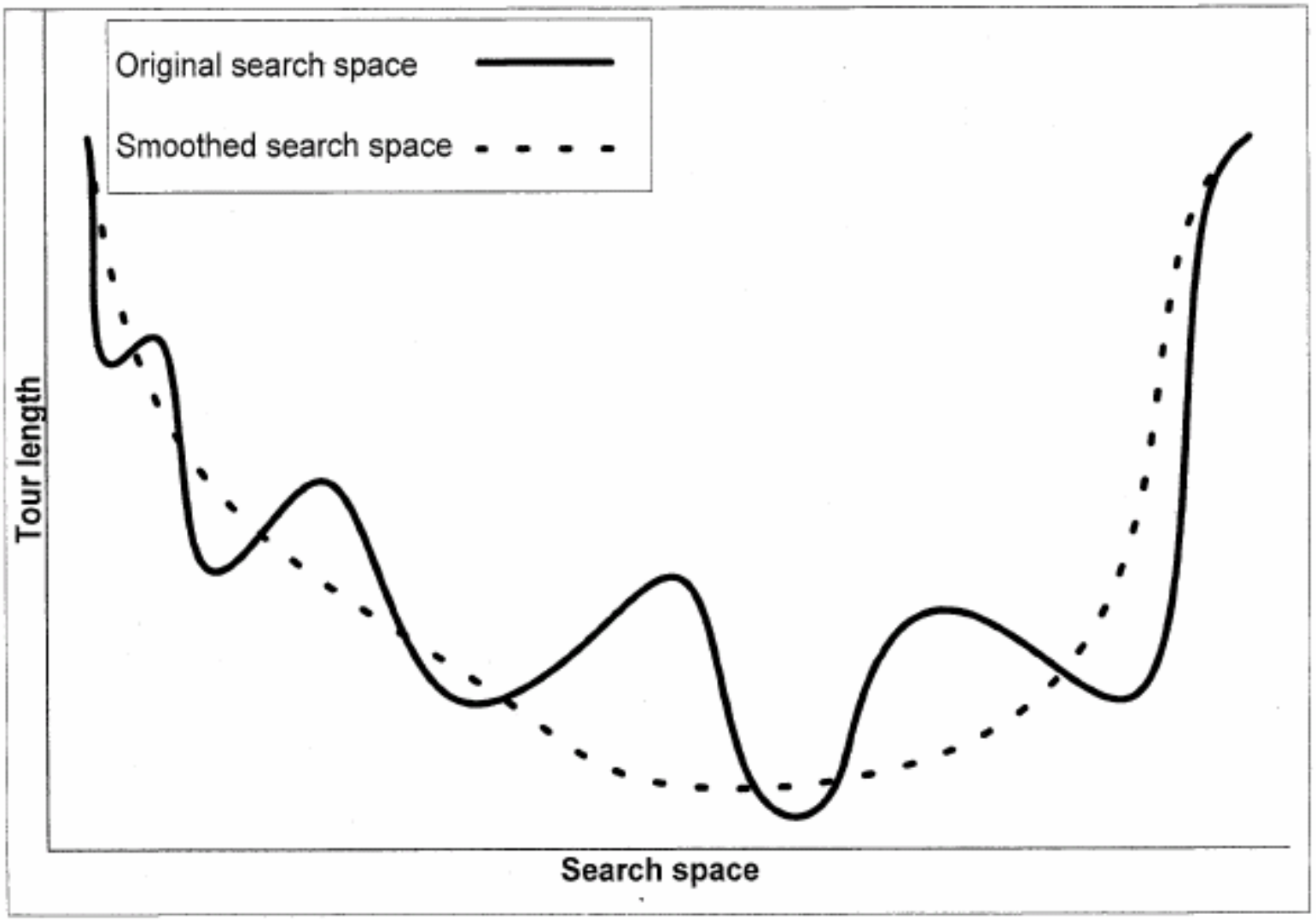
3. If inputs are the original inputs, then stop
   Otherwise, disaggregate or refine the inputs

4. Go to step 2

# Overview of Fine-tuned Learning

- Smoothing and noising are examples of fine-tuned learning
- In this talk, we focus on smoothing and the TSP
- In smoothing, we first normalize distances so that they fall between 0 and 1
- At each iteration, the original distances are transformed by the smoothing transformation. Two-opt is applied. The degree of transformation decreases from one iteration to the next, until the original distances re-emerge.

# Motivation for Search Space Smoothing



# Search Space Smoothing

- Due to Gu and Huang (1994)
- Transform:

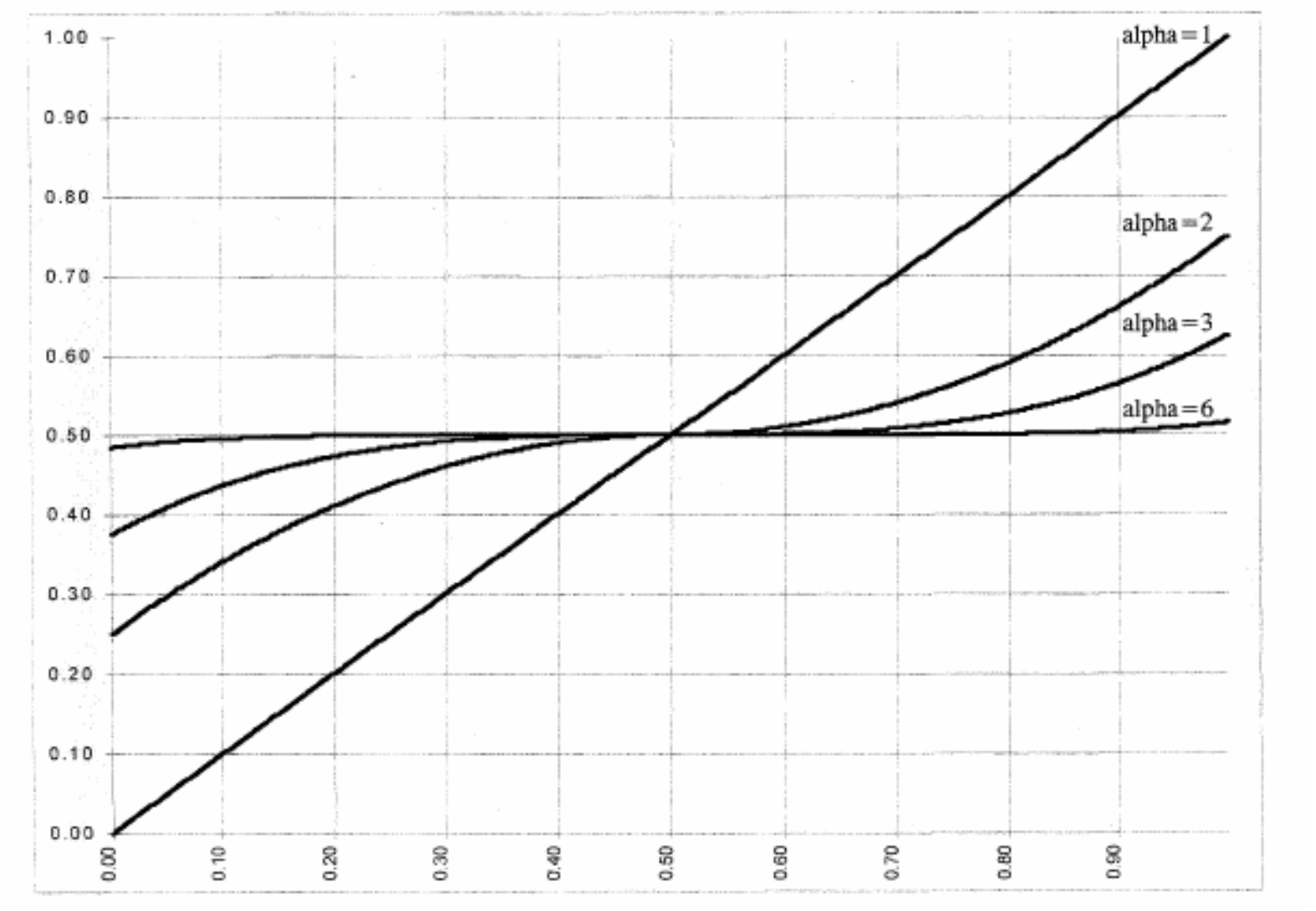
$$d_{ij}(\alpha) = \begin{cases} \bar{d} + (d_{ij} - \bar{d})^{\alpha}, & d_{ij} \geq \bar{d} \\ \bar{d} - (\bar{d} - d_{ij})^{\alpha}, & d_{ij} < \bar{d} \end{cases}$$

Iterations: $\alpha = 6, 3, 2, 1$

This transform clusters distances in the center

The ranking of the distances is preserved

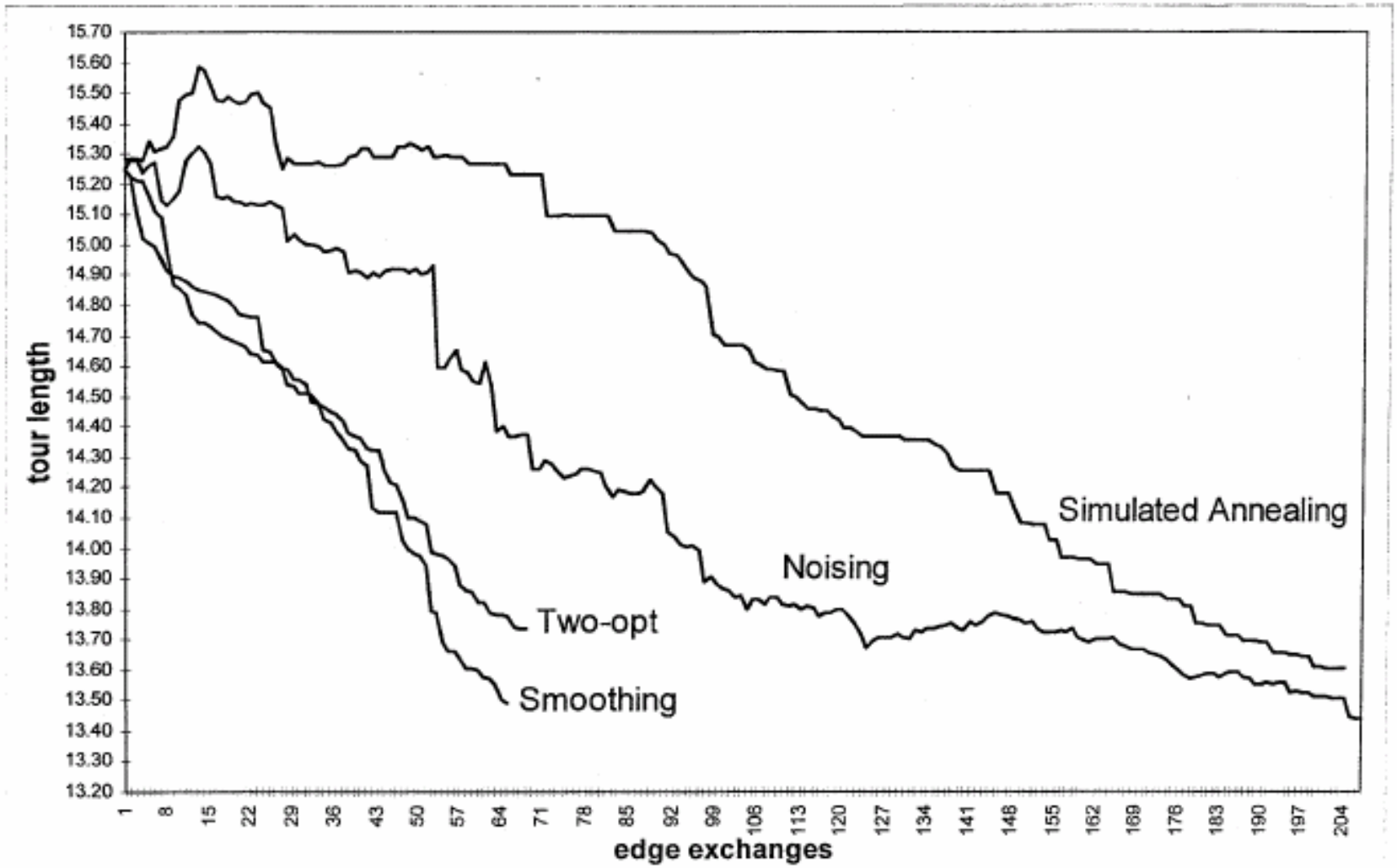
# Smoothing Transformation



# Empirical Results

- Coy, Golden, Runger, and Wasil (1998) studied search-space smoothing in more detail
- Did it work? If so, why?
- We looked at 100, 316, and 1000 node TSPs
  - Euclidean instances and random instances
  - Starting tours were random or came from a greedy heuristic
- Smoothing consistently outperformed two-opt, but was slightly more time-consuming

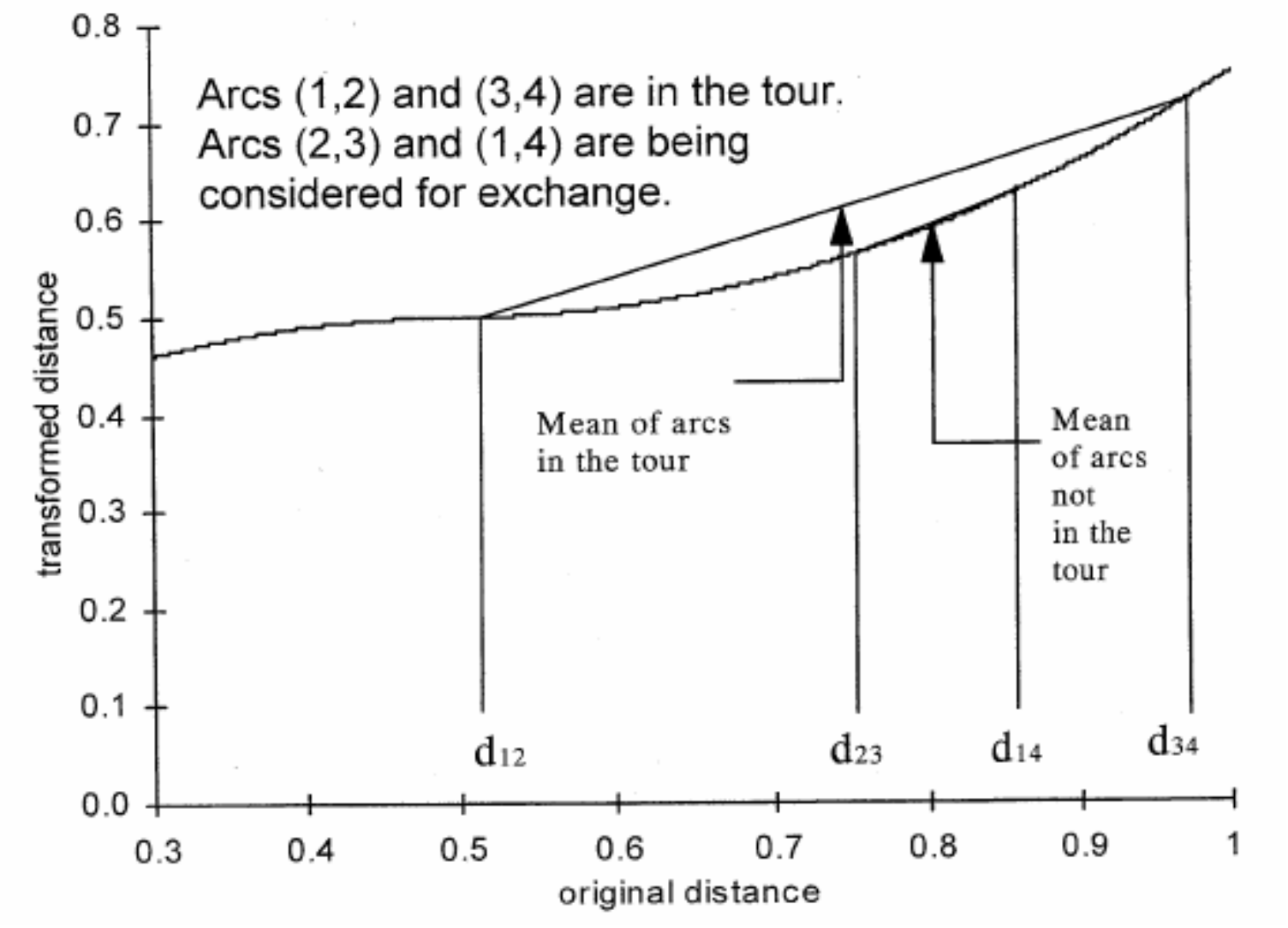
# 316-node Euclidean Instance with a Greedy Start



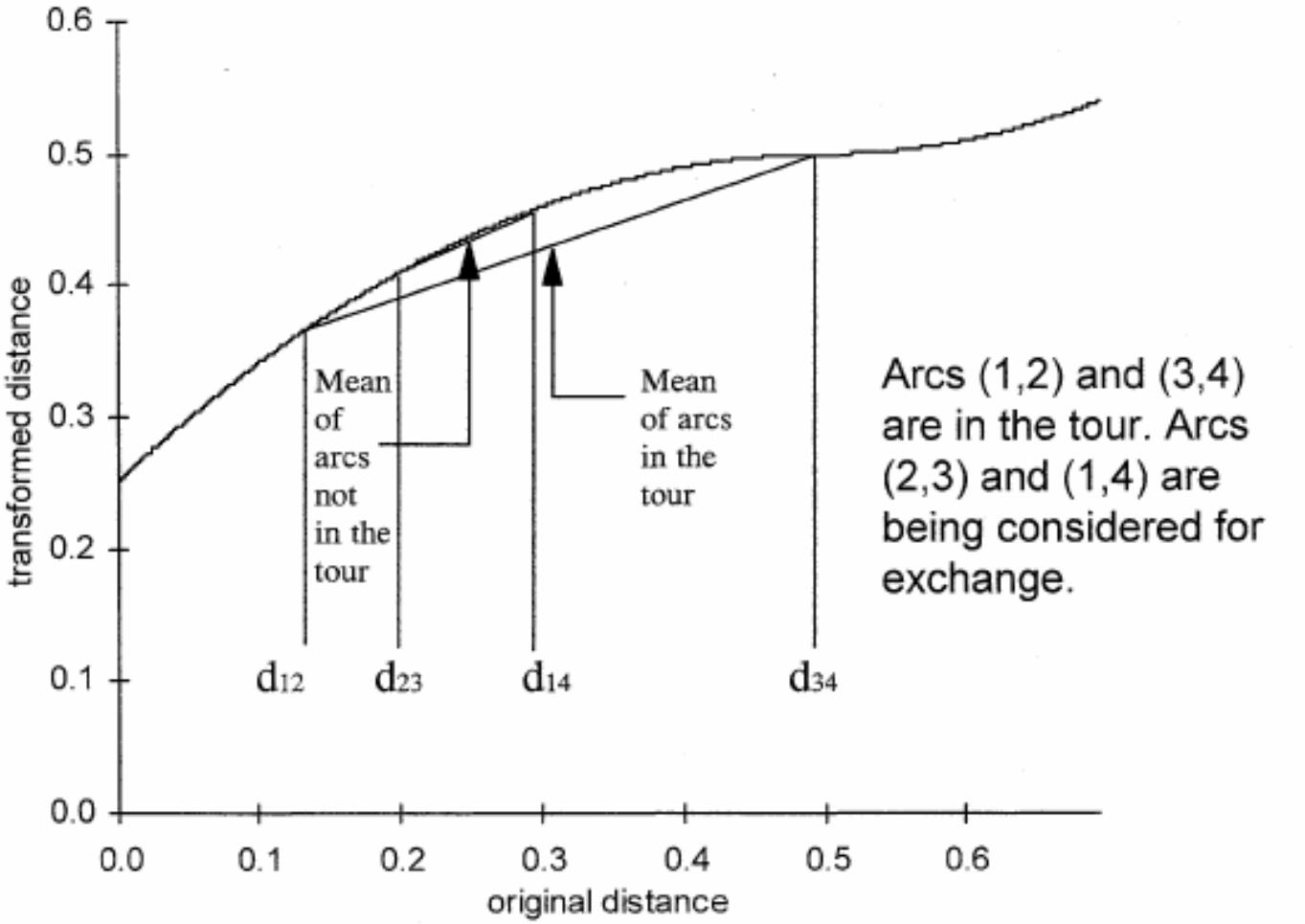
# Why Smoothing Works

- The transition from one transformed distance matrix to the next opens up new opportunities for exchanges
- Smoothing allows some uphill moves
- Smoothing allows the rejection of some marginally improving moves in favor of better moves

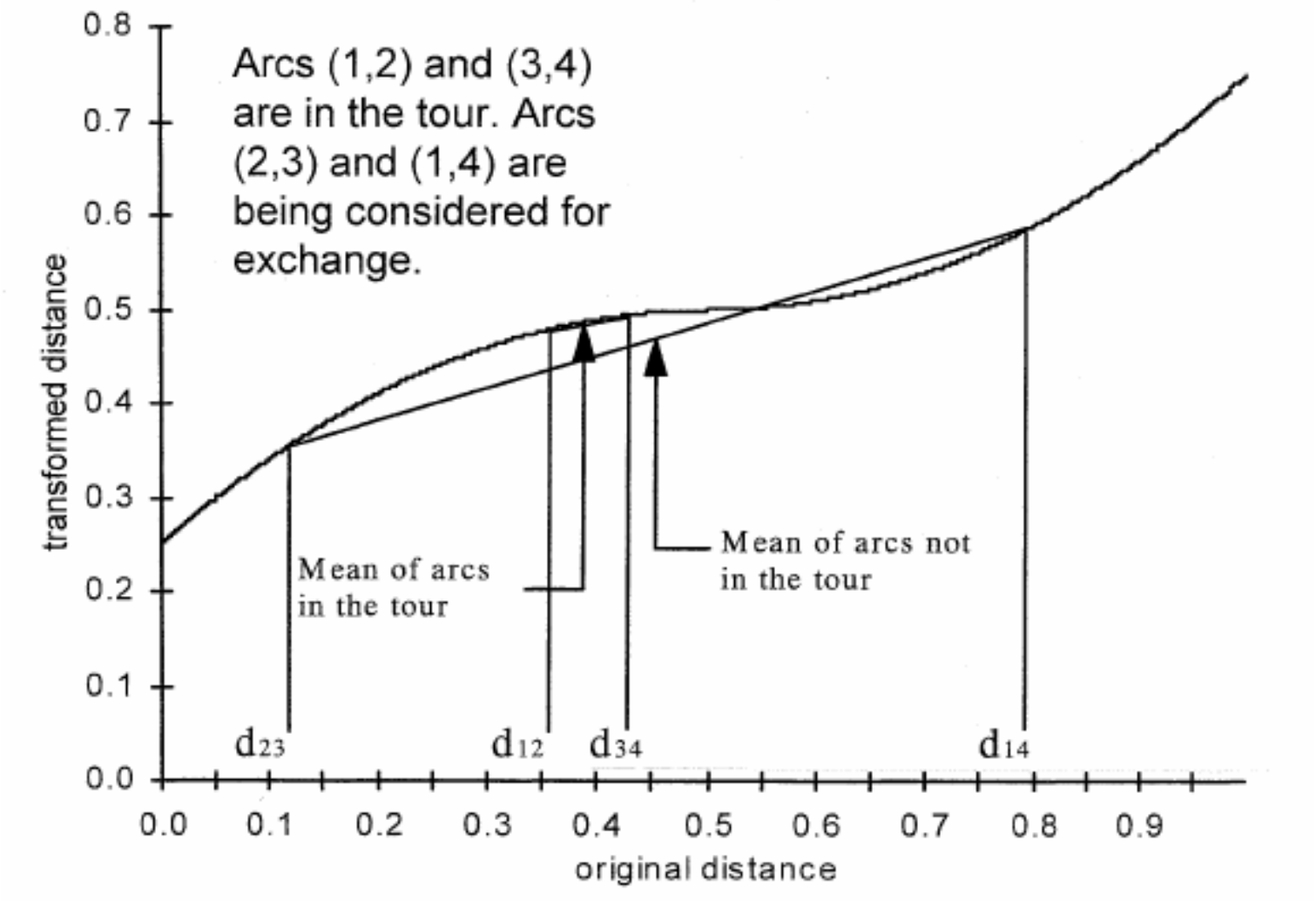
# Unfavorable Moves in the Convex Region of the Transformation



# Marginal Improvements Rejected in the Concave Region of the Transformation



# Unfavorable Moves in the Transitional Region of the Transformation



# Our Work on Smoothing

- We found that smoothing clearly outperforms two-opt and is only slightly more time-consuming
- More importantly, we provided insight into why and how it works
- Smoothing addresses the question

When is a downhill move not what it seems?

# Too Many Parameters!

- Consider a recent paper on vehicle routing
- Published in Transportation Science in the last five years
- It combines a metaheuristic and IP
- There are at least 10 parameters
- How were parameter values set? They don't say
- Do the authors perform sensitivity analysis? No

# Too Many Parameters!

- This paper is typical
- Most metaheuristics papers have too many parameters and the parameter values are not set in a scientific way
- Key observation: The more parameters, the less confidence I have in the metaheuristic
- Suggestions: Researchers should try hard to limit the number of parameters in the metaheuristics they design and provide guidance in setting the values of parameters

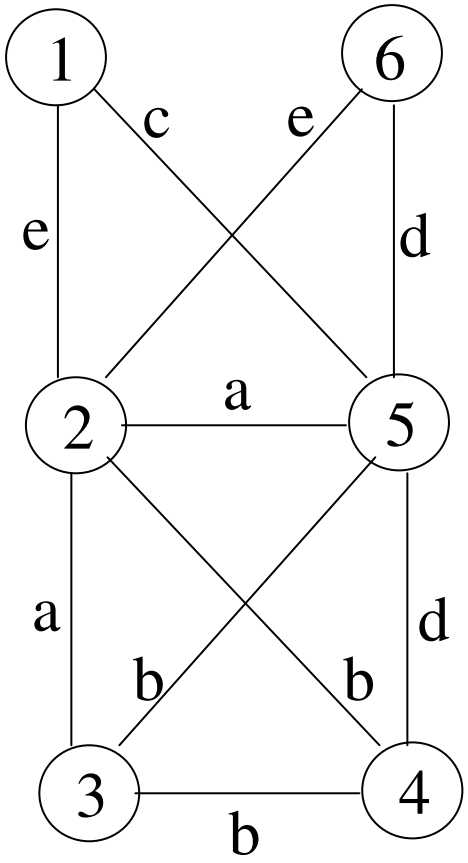
# A One-Parameter Genetic Algorithm

- The minimum label spanning tree (MLST) problem
  - Communications network design
  - Each edge type is denoted by a unique letter or color
  - Construct a spanning tree that minimizes the number of colors
  - See Xiong, Golden, and Wasil (2005, 2006) for details
  - Unlike the MST, where we focus on the edges, here it makes sense to focus on the labels or colors

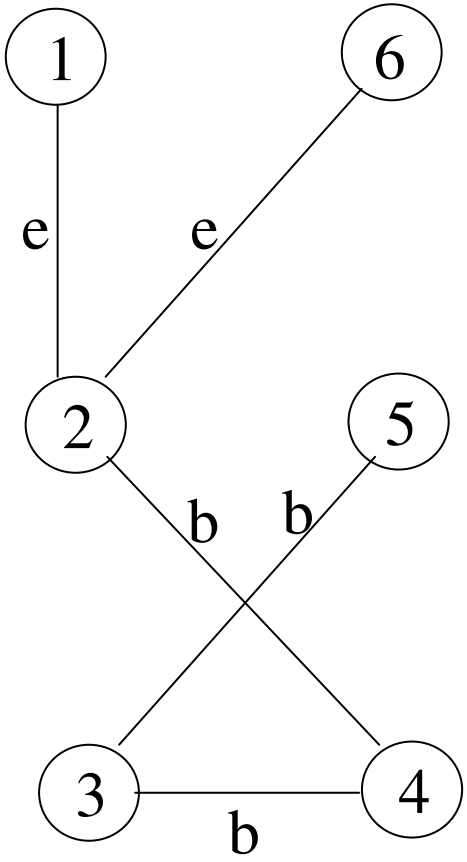
# Introduction

- A Small Example

**Input**




**Solution**



# Genetic Algorithm: Overview

- Randomly choose p solutions to serve as the initial population
- Suppose s [0], s [1], … , s [p – 1] are the individuals (solutions) in generation 0
- Build generation k from generation k – 1 as below

    For each j between 0 and p – 1, do:

        t [ j ] = crossover { s [ j ], s [ (j + k) mod p ] }

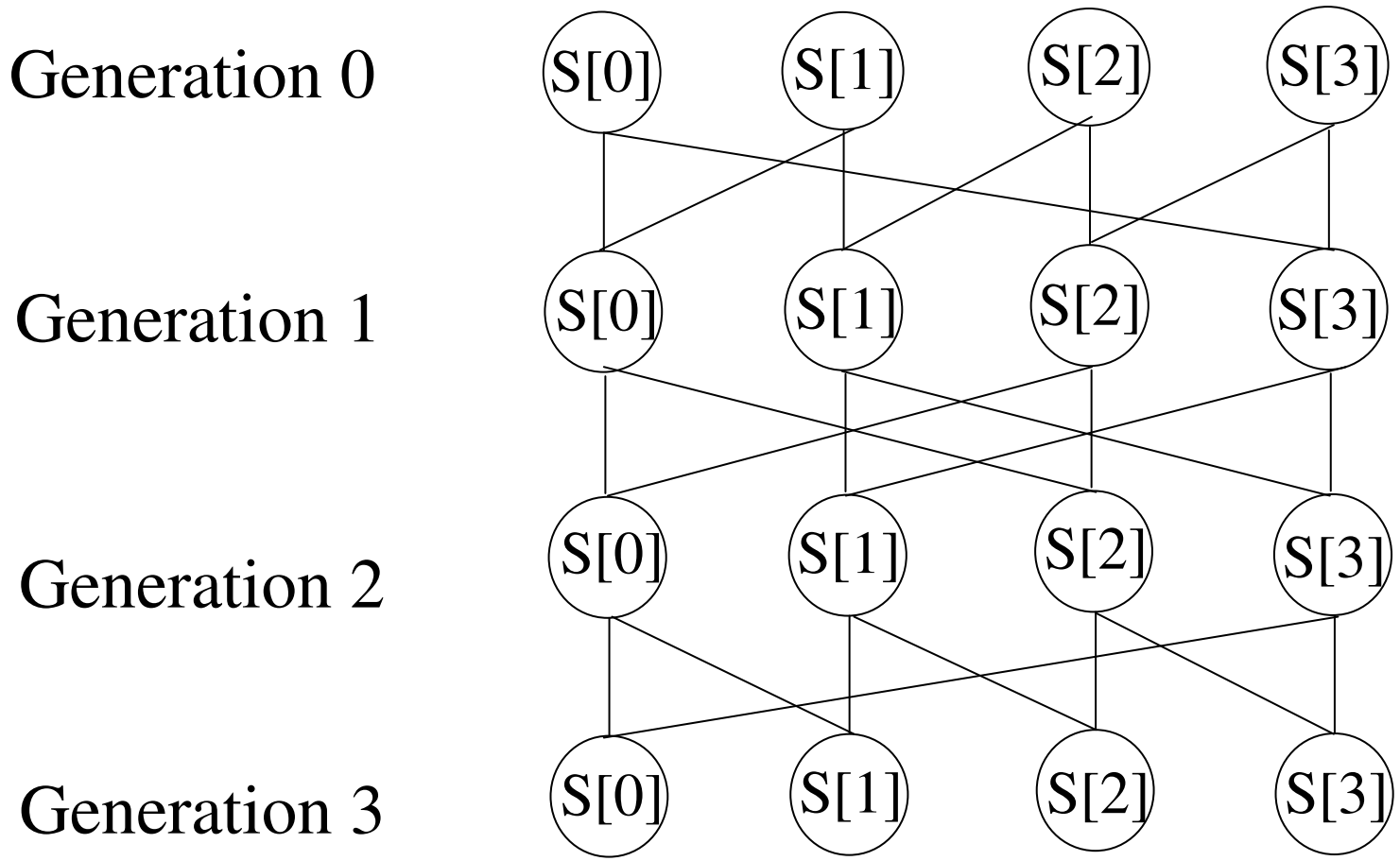
        t [ j ] = mutation { t [ j ] }

        s [ j ] = the better solution of s [ j ] and t [ j ]

    End For

- Run until generation p – 1 and output the best solution from the final generation

# Crossover Schematic (p = 4)



# Crossover

- Given two solutions s [ 1 ] and s [ 2 ], find the child T = crossover { s [ 1 ], s [ 2 ] }
- Define each solution by its labels or colors
- Description of Crossover

    a. Let S = s [ 1 ] $\cup$ s [ 2 ] and T be the empty set

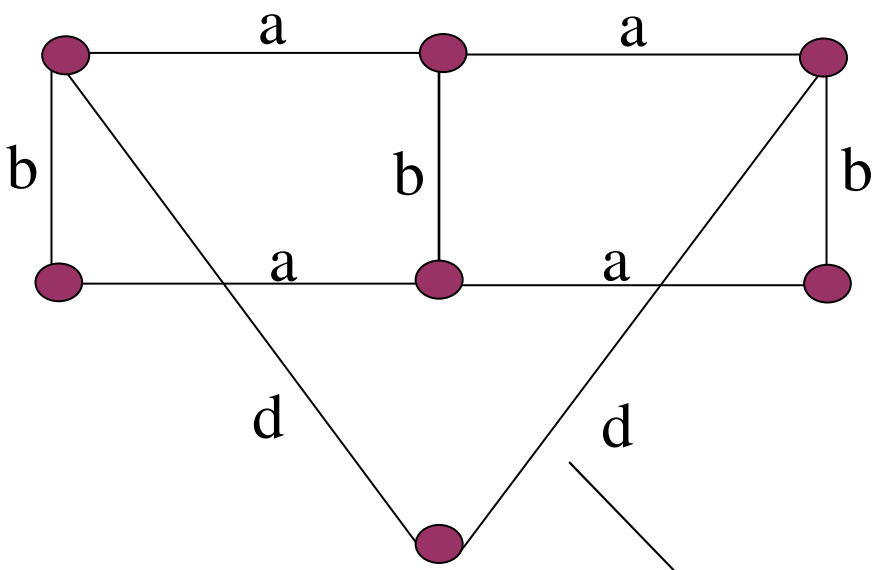
    b. Sort S in decreasing order of the frequency of labels in G

    c. Add labels of S, from the first to the last, to T until T represents a feasible solution

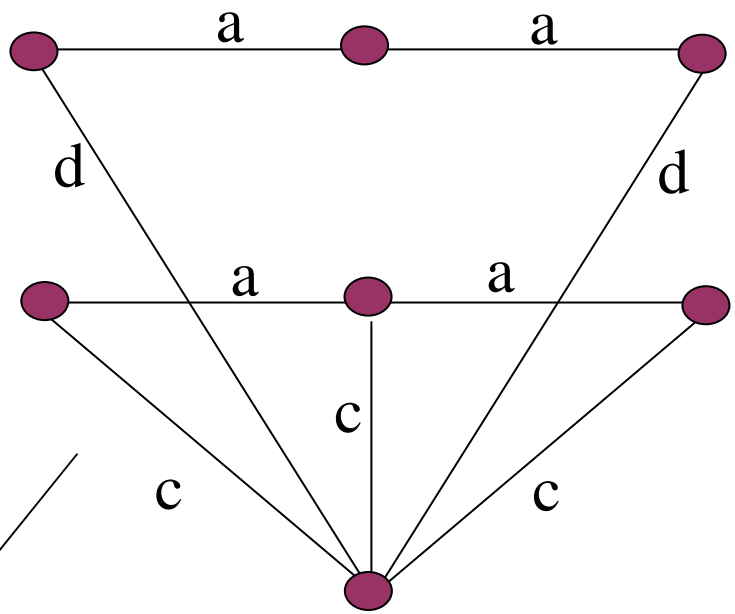
    d. Output T

# An Example of Crossover

s [ 1 ] = { a, b, d }




s [ 2 ] = { a, c, d }



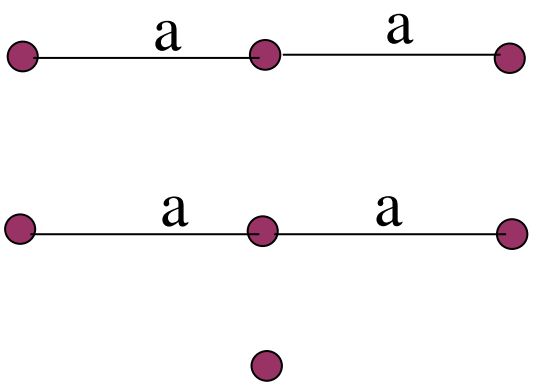

T = { }

S = { a, b, c, d }

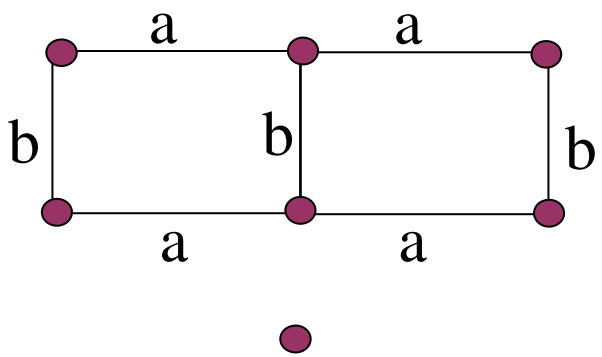
Ordering: a, b, c, d

# An Example of Crossover

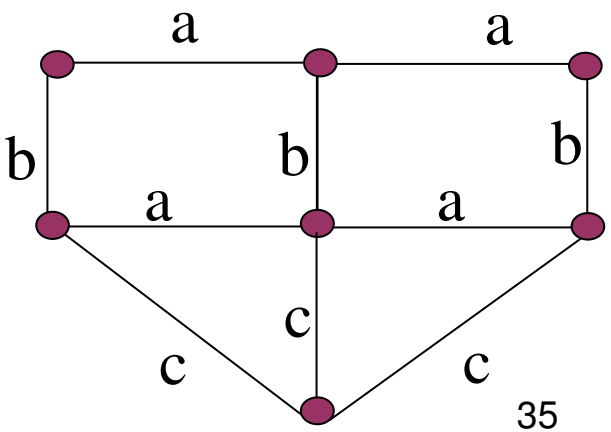
T = { a }




T = { a, b }




T = { a, b, c }



# Mutation

- Given a solution S, find a mutation T
- Description of Mutation

  a. Randomly select c not in S and let T = S $\cup$ c

  b. Sort T in decreasing order of the frequency of the labels in G

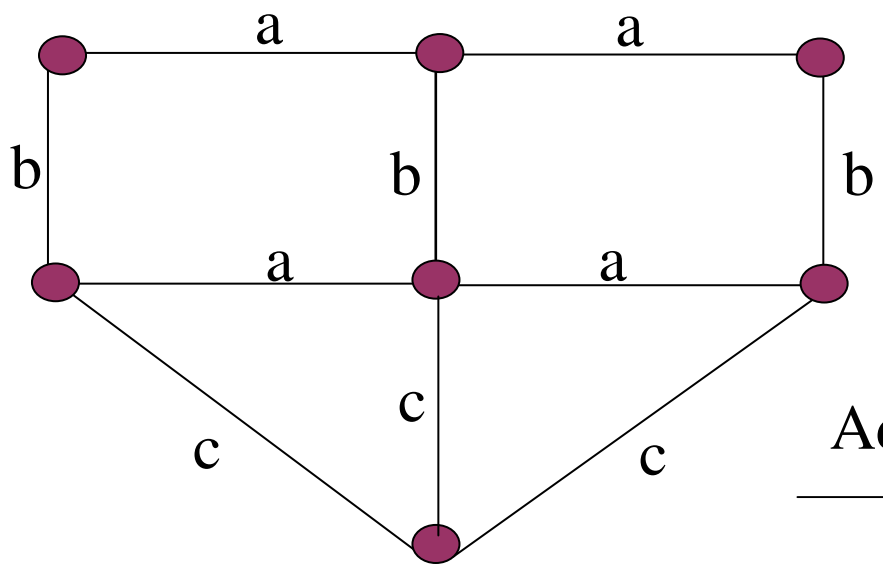
  c. From the last label on the above list to the first, try to remove one label from T and keep T as a feasible solution

  d. Repeat the above step until no labels can be removed

  e. Output T

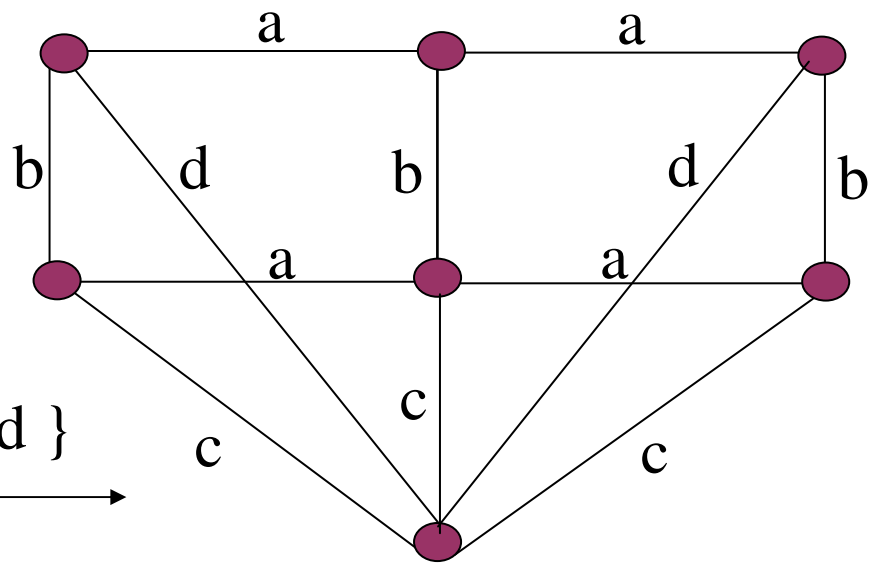
# An Example of Mutation

S = { a, b, c }




Add { d }

S = { a, b, c, d }




Ordering: a, b, c, d

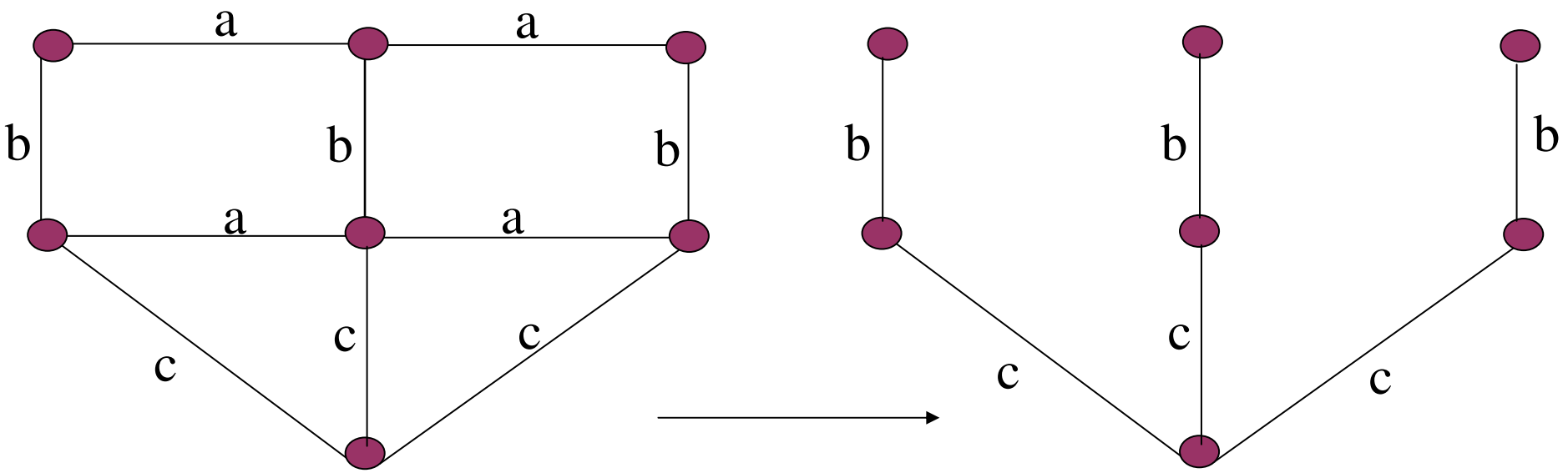
# An Example of Mutation

Remove { d }

S = { a, b, c }

Remove { a }

S = { b, c }




T = { b, c }

# One Parameter is Enough

- Simple experiments confirm that p = 20 or p = 30 works very well
- As expected, for larger problems, slightly better results are obtained with p = 30, but running times are longer
- When there are multiple parameters
  - "Train" on a small subset of benchmark problems
  - The parameter values are then set
  - Next, "test" the metaheuristic on the remaining benchmark problems

# Some Interesting Open Research Questions

- How do we determine/recognize the right metaheuristic for a problem class (e.g., bin packing or the TSP)?
- Why does a metaheuristic work well for a problem class?
- More specifically, on what types of problem instances (e.g., random vs. clustered TSPs) does a metaheuristic work well? When does it perform poorly?

# Research Questions and a Conclusion

- How do we design simple and robust hybrid algorithms (e.g., combining metaheuristics and IP) for solving combinatorial optimization problems?
- Can we begin to apply parallel computation to the design and testing of metaheuristics on a widespread scale?
- Conclusion: We have come a long way, but there are still many important questions for us to answer