Conference on
Numerical Optimization and Applications in Engineering

13 – 15 October 2010

Centre de Recerca Matemàtica · Bellaterra, Barcelona

Using Multi-Start Randomized Heuristics to solve Non-Smooth and Non-Convex Optimization Problems

UOC www.uoc.edu

MIT Massachusetts Institute of Technology

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa

UPC

A. Juan, D. Ionescu, J. Faulin, A. Ferrer

ajuanp@gmail.com | http://ajuanp.wordpress.com

Computer Science Department

Open University of Catalonia, SPAIN

0. Introduction (1/2)

- Combinatorial optimization problems have posed numerous challenges throughout the past decades. They have a well-structured definition consisting of an objective function that needs to be minimized or maximized and a series of constraints.
- The main reason for which they have been so actively investigated is the tremendous amount of real-life applications that can be successfully modeled in this way (e.g.: routing/scheduling issues).
- In some cases, the solution space can be easily explored due to certain properties of the functions involved, such as convexity. However, in other (most?) circumstances, the solution space is highly irregular and finding the optimum is quite difficult.

$$\begin{aligned} \min_{t,\mathbf{a}\neq 0} \quad & t - \mathbf{a}^T(\overline{\mathbf{x}} - \overline{\mathbf{y}}) \\ s.t. \quad & \| \Sigma_{\mathbf{x}}^{\frac{1}{2}} \mathbf{a} \| \leq 1, \\ & \| \Sigma_{\mathbf{y}}^{\frac{1}{2}} \mathbf{a} \| \leq \sqrt{\frac{1-\beta_0}{\beta_0}} t \end{aligned}$$

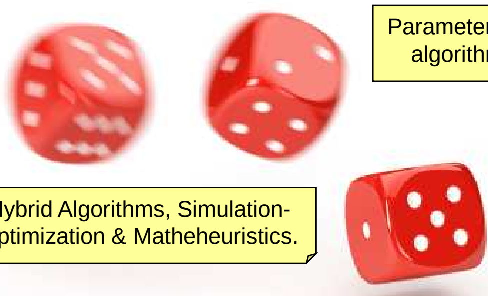
Combinatorial Optimization Problems

Real-life objective functions and constraints are complex!

0. Introduction (2/2)

- Simplicity, efficiency, robustness and flexibility are the attributes that can make one approach better or more suitable than another.
- We propose an alternative non-uniform (or biased) randomization approach that can be easily applied to a variety of non-smooth and/or non-convex optimization problems.
- Basically, our method pertains to the class of nondeterministic or stochastic methods and relies on random sampling. Therefore, on different runs of the algorithm we get different good solutions.
- Having a pool of solutions to choose from can be especially useful in real-life problems when the best known solution may be unfeasible or inappropriate due to external constraints or strange consumer preferences (utility function).

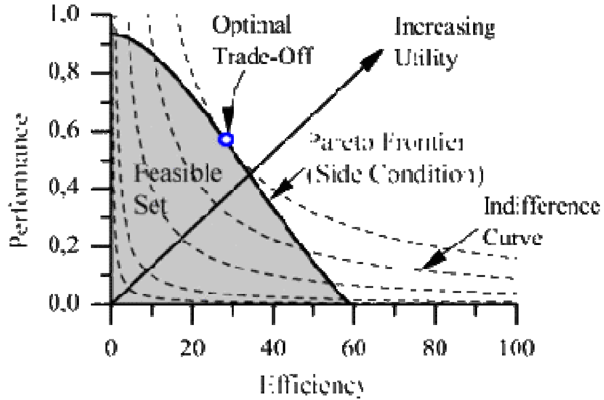
Parameter-free algorithms



Hybrid Algorithms, Simulation-Optimization & Matheuristics.

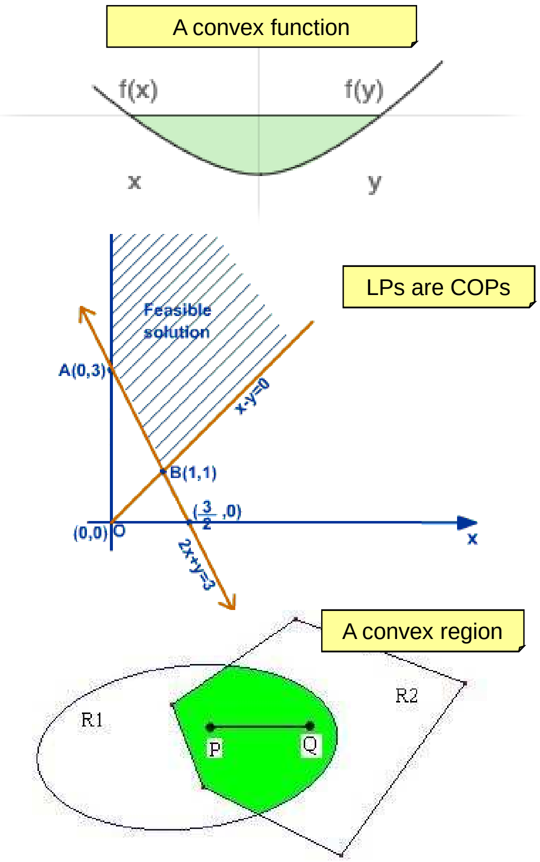
Real-life is usually stochastic (stochastic demands, stochastic times, etc.)

Utility Function & Pareto Frontier



1. Convex Optimization Problems (COPs)

- COPs are problems where all of the constraints are convex functions, and the objective is a convex function if minimizing, or a concave function if maximizing.
- Linear functions are convex, so LP problems are COPs.
- In a COP, the feasible region –the intersection of convex constraint functions– is also a convex region.
- With a convex objective and a convex feasible region, there can be only one optimal solution, which is globally optimal. Several methods –e.g. Interior Point methods– will either find the globally optimal solution, or prove that there is no feasible solution to the problem.
- COPs can be solved efficiently up to very large size.

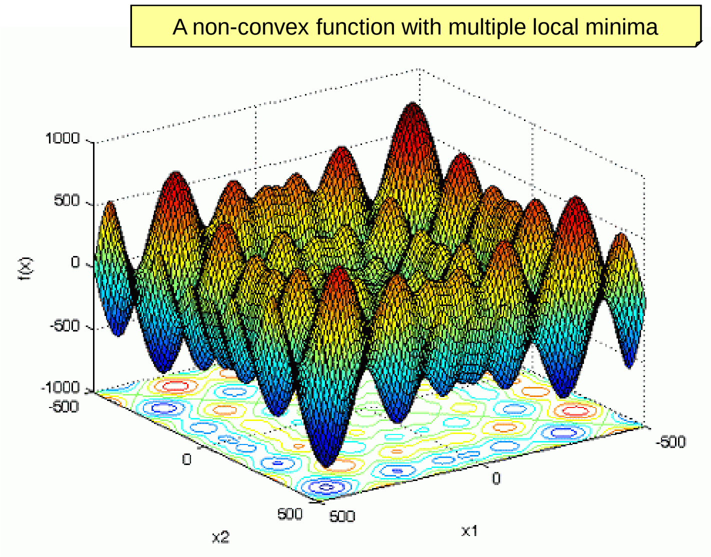


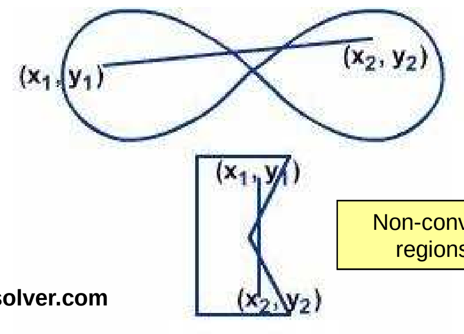
Source: www.solver.com

2. Non-convex Optimization Problems (NCOPs)

- NCOPs are problems where either the objective or any of the constraints are non-convex.
- NCOPs may have multiple feasible regions and multiple locally optimal points within each region.
- It can take exponential time in the number of variables and constraints to determine that a non-convex problem is infeasible, that the objective function is unbounded, or that an optimal solution is the "global optimum" across all feasible regions.

A non-convex function with multiple local minima





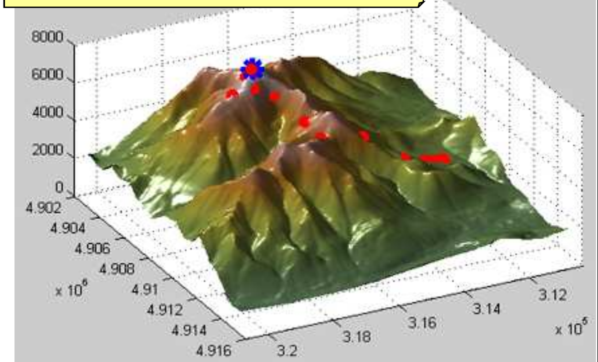
Non-convex regions

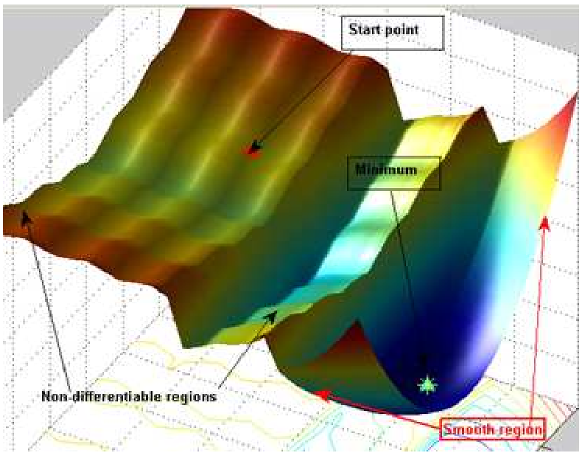
Source: www.solver.com

3. Non-Smooth Optimization Problems (NSPs)

- Typically, NSPs are also NCOPs. Hence:
  - They might have multiple feasible regions and multiple locally optimal points within each region –because some of the functions are non-smooth or even discontinuous, and
  - Derivative/gradient information generally cannot be used to determine the direction in which the function is increasing (or decreasing).
- In a NSP, the situation at one possible solution gives very little information about where to look for a better solution.
- In most NSPs it is impractical to enumerate all of the possible solutions and pick the best one. Hence, most methods rely on some sort of random sampling of possible solutions.
- Such methods are nondeterministic or stochastic –they may yield different solutions on different runs, depending on which points are randomly sampled.

Examples of non-smooth functions



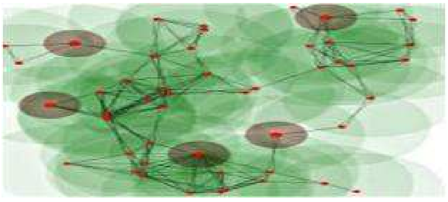


Source: www.mathworks.com

3. Examples of Nonsmooth Functions

The Problem	The objective function

- **Sensor Network Localization**

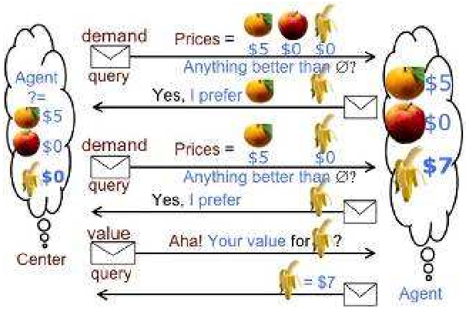


$$\sum_{(i,j)\in N_x} \max\left(0, -\|x^i - x^j\|^2 + \underline{d}_{ij}^2, \|x^i - x^j\|^2 - \overline{d}_{ij}^2\right) + \sum_{(i,k)\in N_a} \max\left(0, -\|x^i - a^k\|^2 + \underline{d}_{ik}^2, \|x^i - a^k\|^2 - \overline{d}_{ik}^2\right).$$

- **Optimal Circuit Routing**

$$\max_H F(H) = \min_i \left(V_i - \sum_{k=1}^{K} \sum_{s=1}^{p(k)} P_s^k(i) h_s^k\right)$$

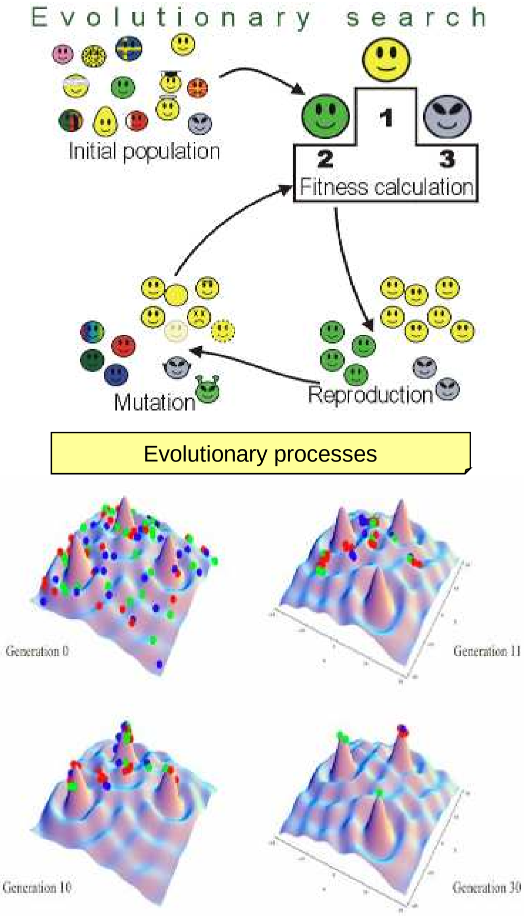
- **Winner Determination Problem (combinatorial auctions)**



$$\max \sum_{j\in J} \sum_{S\subseteq I} b^j(S)\, x(S,j)$$

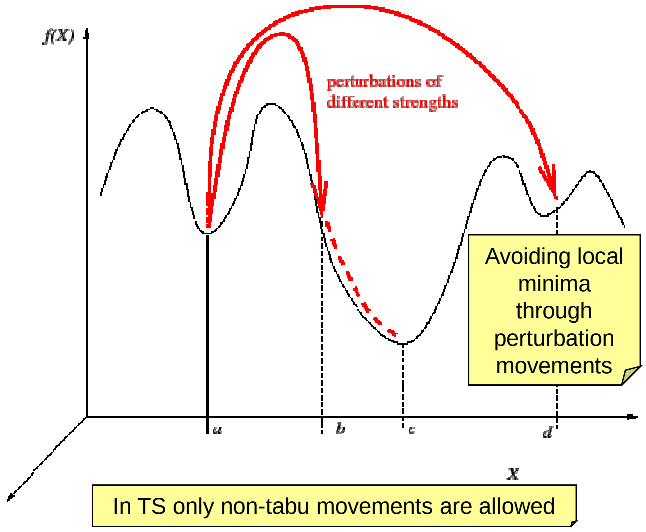
4. Solving NSPs with GAs and EAs

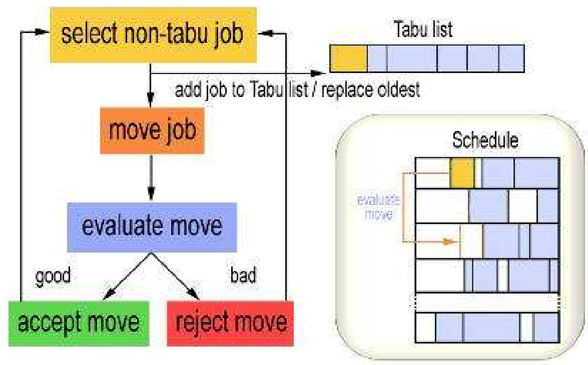
- Genetic and Evolutionary Algorithms offer one way to find "good" solutions to non-smooth optimization problems:
  - In a genetic algorithm the problem is encoded in a series of bit strings that are manipulated by the algorithm.
  - In an "evolutionary algorithm," the decision variables and problem functions are used directly.
- GAs and EAs maintain a population of candidate solutions, rather than a single best solution so far. From existing candidate solutions, they generate new solutions through either random mutation of single points or crossover or recombination of two or more existing points. The population is then subject to selection that tends to eliminate the worst candidate solutions and keep the best ones. This process is repeated, generating better and better solutions.
- There is no way for these methods to determine that a given solution is truly optimal.



5. Solving NSPs with Tabu Search

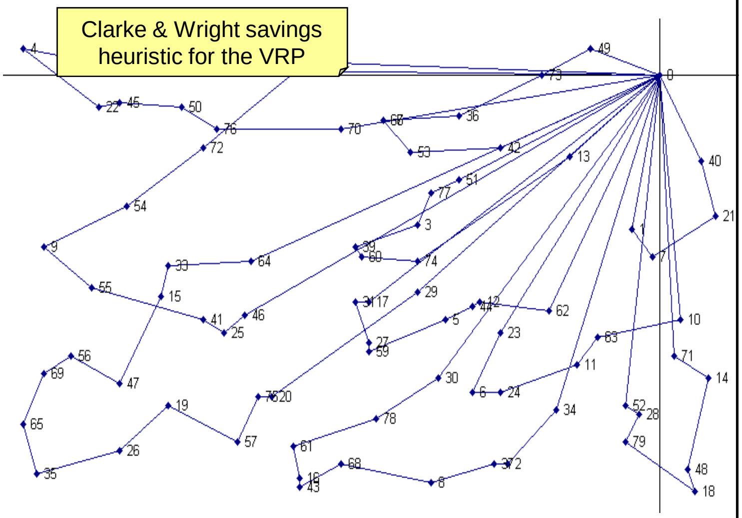
- Tabu Search algorithms offer another approach to find "good" solutions to non-smooth optimization problems.
- TS algorithms also maintain a population of candidate solutions, rather than a single best solution so far, and they generate new solutions from old ones. However, they rely less on random selection and more on deterministic methods.
- Tabu search uses memory of past search results to guide the direction and intensity of future searches.
- These methods generate successively better solutions, but as with genetic and evolutionary algorithms, there is no way for these methods to determine that a given solution is truly optimal.
- Other approaches exist: GRASP, ACO, SA, etc.

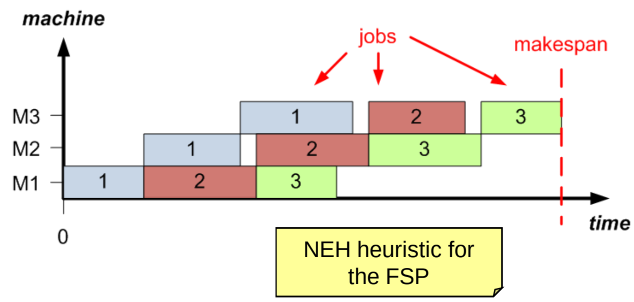




6. Randomizing Classical Heuristics (1/2)

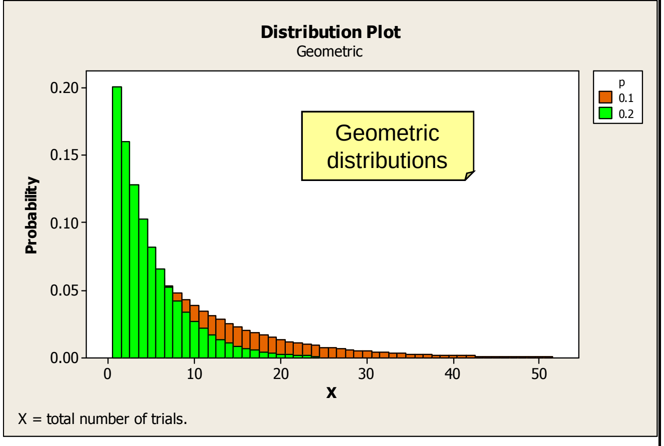
- There are excellent and well-tested classical constructive heuristics for almost every relevant combinatorial optimization problem (e.g.: vehicle routing problem, scheduling problems, allocation problems, etc.).
- Being constructive methods, these heuristics tend to perform well even in the case of non-smooth functions and non-convex functions and regions.
- During the constructive stage, these heuristics select the next movement, from a list of available movements, according to a greedy criterion, e.g.: “select the node which the highest savings” (CWS for the VRP) or “select the job with the highest processing time” (NEH for the FSP), etc.

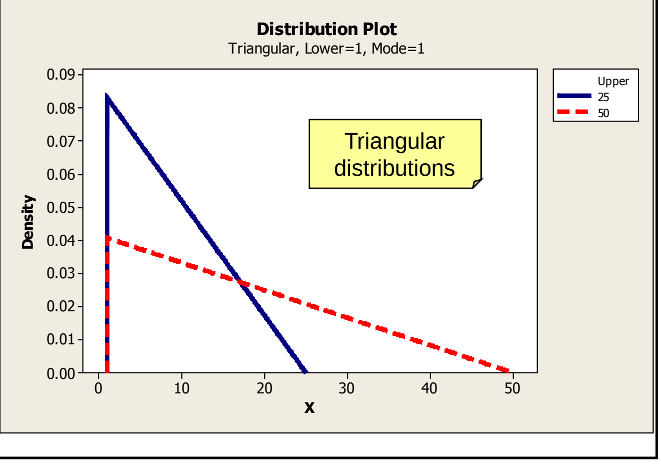




6. Randomizing Classical Heuristics (2/2)

- We propose to introduce a biased random behavior (BRB) in these selection process so that movements with better values have higher probabilities of being selected, but other movements could also be selected instead at each step.
- This way, deterministic classical heuristics (e.g.: Clarke and Wright, NEH, etc.) are transformed into probabilistic ones without losing the "common sense" rules that make them efficient.
- Thus, we transform a "gun heuristic" into a "machine-gun heuristic": each time the randomized heuristic is run, a different "good" solution will be obtained (kind of a "Biased GRASP").
- The geometric and the discrete version of the triangular can be used to infer this BRB.

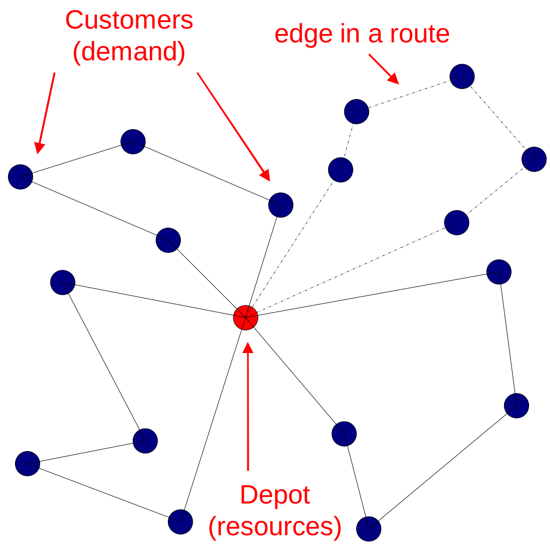




7. Example 1: The CVRP (1/8)

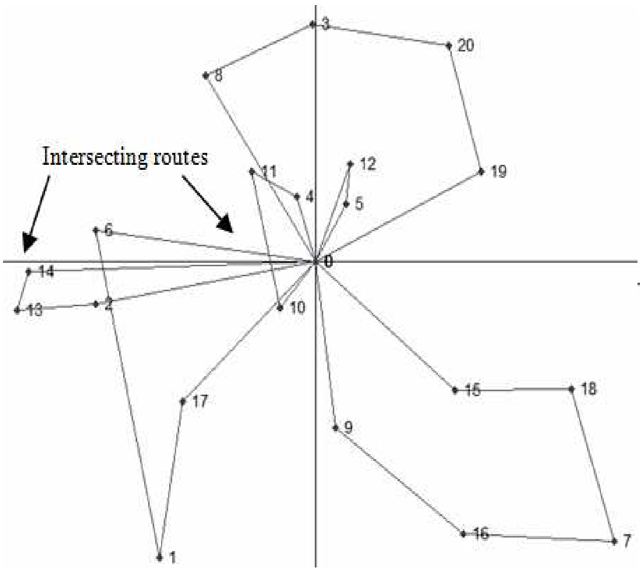
- The Vehicle Routing Problem (CVRP) is a well-known NP-hard problem:
  - A set of customers' demands must be supplied by a fleet of vehicles.
  - Resources are available from a depot.
  - Moving a vehicle from one node i to another j has associated costs $c(i, j)$
  - Several constraints must be considered: maximum load capacity per vehicle, service times, etc.

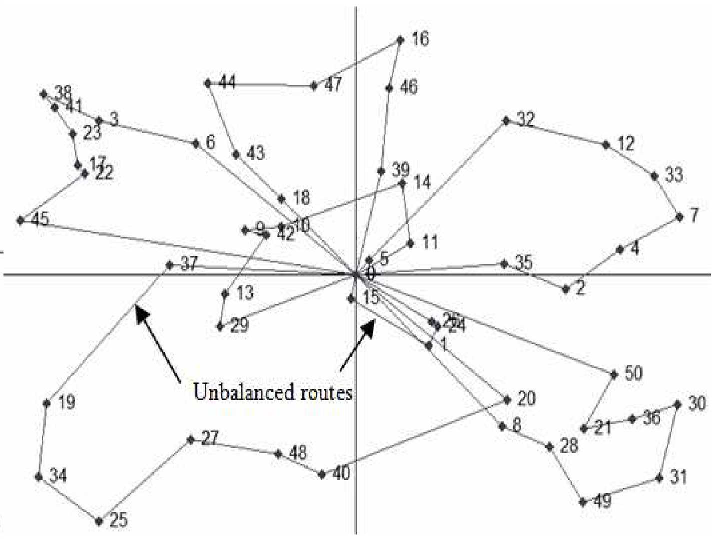
- Goal: to obtain an optimal solution, i.e. a set of routes satisfying all constraints with minimum costs
- Different approaches for CVRP: optimization methods (small-size), heuristics (CWS) and meta-heuristics (GAs, TS, SA, GRASP, …)



7. Example 1: The CVRP (2/8)

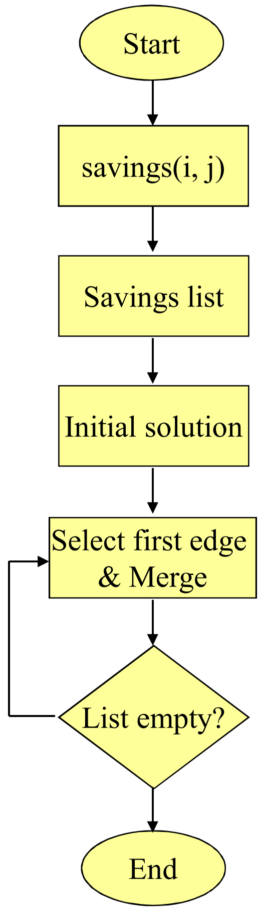
- Yes, but…: In real-life scenarios is not possible to model all costs, constraints and desirable solution properties in advance (Kant et al 2008)
- Goal 2 (our approach): to develop a method that <u>also</u> provides many ‘good’ alternative solutions, so that the decision-maker can select the one that best fits her utility function.





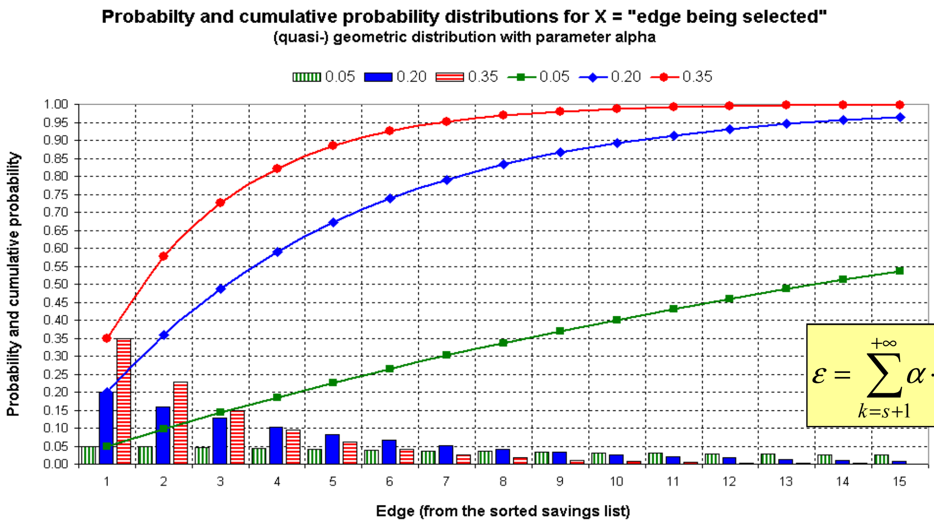
7. Example 1: The CVRP (3/8)

- Our approach will be based on the Clarke and Wright's savings (CWS) algorithm (Clarke & Wright 1964).
- CWS algorithm:
  1. For each pair of nodes i and j, calculate the savings, $s(i, j)$, associated to the edge connecting them, where: $s(i, j) = c(0, i) + c(0, j) - c(i, j)$
  2. Construct a list of edges, sorting the edges according to their associated savings
  3. Construct an initial feasible solution by routing a vehicle to each client node
  4. Select the first edge in the savings list and, if no constraint is violated, merge the routes that it connects
  5. Repeat step 4 until the savings list is empty
- This parallel version of the CWS heuristic usually provides 'acceptable solutions' (average gap between 5% and 10%), especially for small and medium-size problems



7. Example 1: The CVRP (4/8)

- CWS → the <u>first edge</u> (the one with the most savings) is the one selected.
- SR-GCWS introduces randomness in this process by using a quasi-geometric statistical distribution → edges with more savings will be more likely to be selected at each step, but all edges in the list are potentially eligible.
- Notice: Each time SR-GCWS is run, a random feasible solution is obtained. By construction, chances are that this solution outperforms the CWS one → hundreds of ‘good’ solutions can be obtained after some seconds/minutes.



Good results with $0.10 < \alpha < 0.20$

$$\forall k = 1,2,\ldots,s$$

$$P(X = k) = \alpha \cdot (1-\alpha)^{k-1} + \varepsilon$$

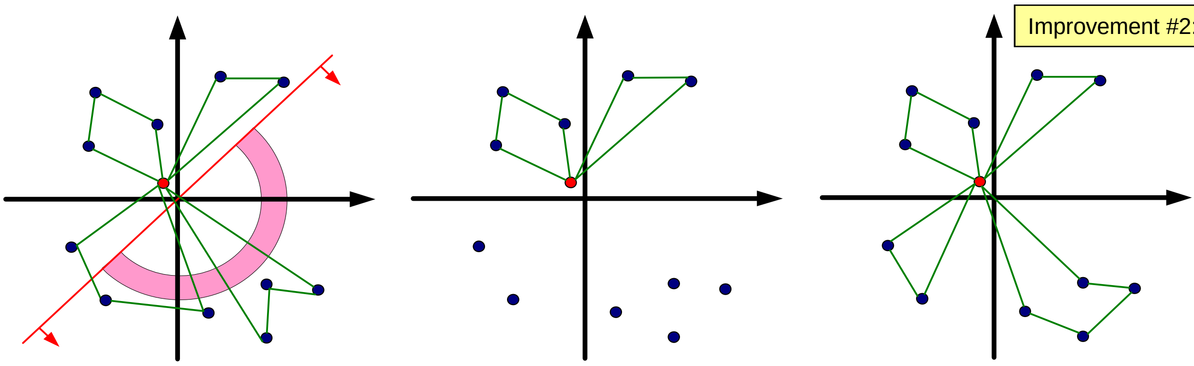
$$\varepsilon = \sum_{k=s+1}^{+\infty} \alpha \cdot (1-\alpha)^{k-1} = 1 - \sum_{k=1}^{s} \alpha \cdot (1-\alpha)^{k-1}$$

7. Example 1: The CVRP (5/8)

Improvement #1: **Hash Table**

1. Adding ‘memory’ to our algorithm with a hash table:
   - A hash table is used to save, for each generated route, the best-known sequence of nodes (this will be used to improve new solutions)
   - ‘Fast’ method that provides small improvements on the average
2. Splitting (divide-and-conquer) method:
   - Given a global solution, the instance is sub-divided in smaller instances and then the algorithm is applied on each of these smaller instances
   - ‘Slow’ method that can provide significant improvements

Improvement #2: **Splitting**



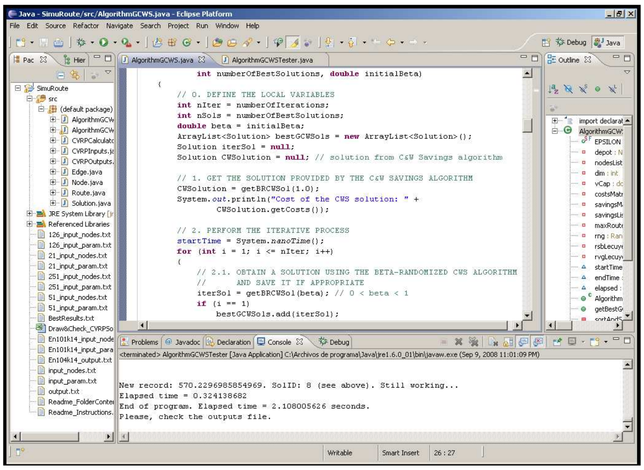
1. Select routes on the SE area (area below the diagonal)

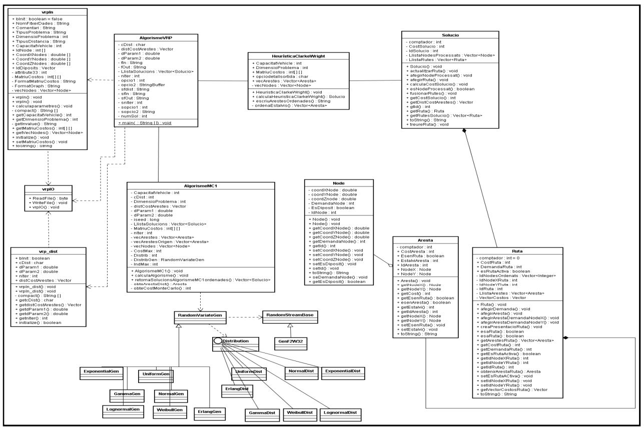
2. Consider the new CVRP subproblem

3. Solve the subproblem and re-construct the solution

7. Example 1: The CVRP (6/8)

- OO approach (Java, Eclipse)
- Special attention:
  i. RNG (L'Ecuyer 2001) → SSJ library (L'Ecuyer 2002), GenF2W32 period 2^{800}-1
  ii. Design of classes (Horstmann 2006)
  iii. Code accuracy and effectiveness
- Implementation of the CWS heuristic (parallel version) based on: http://web.mit.edu/urban_or_book/www/book/chapter6/6.4.12.htm





Both the CWS and the SR-GCWS-CS implementations have been **verified** by using standard benchmarks and independent calculations

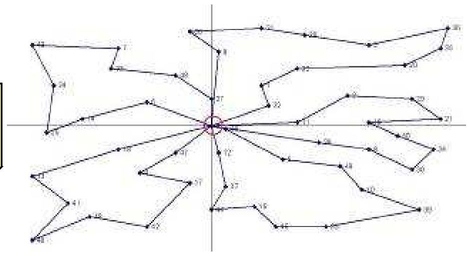
7. Example 1: The CVRP (7/8)

- To verify the goodness of our approach and its efficiency, a total of 50 classical VRP benchmark instances were randomly selected from http://www.branchandcut.org (which also contains best-known solutions so far)

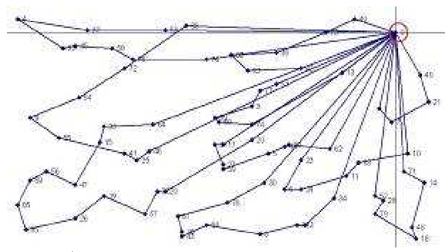
- Results:
  a) 31-out-of-50 instances offer a negative gap –i.e., they outperform the BKS
  b) The remaining 19 instances offer a null gap
  c) Average gap = -0.21%
  d) In most cases → few seconds

Different Scenarios:
• From 45 to 200 nodes
• Different topologies (depot, clusters, etc.)

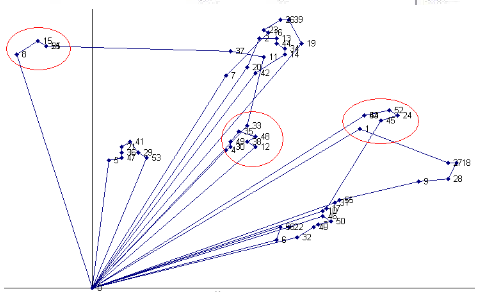
E-n51-k5.vrp
Depot at the center



A-n80-k10.vrp
Depot at one corner



B-n57-k9.vrp
Cluster topology



7. Example 1: The CVRP (8/8)

Intel® Core™2 Duo CPU at 2.4 GHz and 2 GB RAM

A **positive gap** implies that the CWS solution costs are higher than the ones associated with the best-known-so-far solution.

Table 1. Comparison of methodologies for the fifteen selected CVRP instances

Instance	Nodes	CWS-p solution (1)	Gap (1) – (2)	Best-known solution* (2)	SR-GCWS solution (3)	Gap (2) – (3)
A-n45-k7	45	1,199.98	4.59%	1,147.28	1,146.91	-0.03%
A-n60-k9	60	1,421.88	4.87%	1,355.80	1,355.80	0.00%
A-n80-k10	80	1,860.94	5.35%	1,766.50	1,766.50	0.00%
B-n50-k7	50	748.80	0.54%	744.78	744.23	-0.07%
B-n52-k7	52	764.90	1.98%	750.08	749.97	-0.01%
B-n57-k9	57	1,653.42	3.10%	1,603.63	1,602.29	-0.08%
B-n78-k10	78	1,264.56	2.87%	1,229.27	1,228.16	-0.09%
E-n51-k5	51	584.64	11.37%	524.94	524.61	-0.06%
E-n76-k10	76	900.26	7.51%	837.36	839.13	0.21%
E-n76-k14**	76	1,073.43	4.55%	1,026.71	1,026.14	-0.06%
F-n135-k7	135	1,219.32	4.16%	1,170.65	1,170.33	-0.03%
M-n121-k7	121	1,068.14	2.20%	1,045.16	1,045.60	0.04%
M-n200-k7	200	1,395.74	6.10%	1,315.43	1,313.71	-0.13%
P-n70-k10	70	896.86	10.56%	830.02	831.81	0.22%
P-n101-k4	101	765.38	8.05%	692.28	691.29	-0.14%
Averages			*5.19%*			*-0.02%*

(*) Best-known solution according to the information available at http://www.branchandcut.org/

A **negative gap** implies that our solution costs are lower than the ones associated with the best-known-so-far solution.

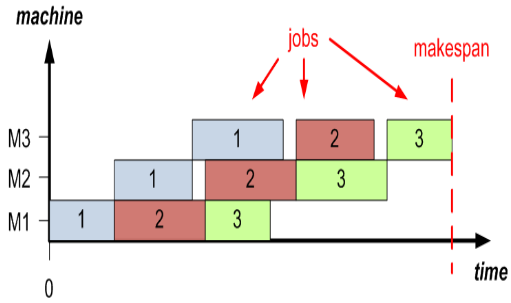
Juan, A.; Faulin, J.; Ruiz, R.; Barrios, B.; Caballe, S. (2010): "The SR-GCWS hybrid algorithm for solving the capacitated vehicle routing problem". *Applied Soft Computing*, Vol. 10, No. 1, pp. 215-224

Juan, A.; Faulin, J.; Jorba, J.; Riera, D.; Masip; D.; Barrios, B. (2010): "On the Use of Monte Carlo Simulation, Cache and Splitting Techniques to Improve the Clarke and Wright Savings Heuristics". Journal of the Operational Research Society. doi:10.1057/jors.2010.29

Our approach improves **31-out-of-50** benchmark solutions, with a global **average gap of -0.21%** for the 50 benchmark instances

8. Example 2: The FSP (1/4)

- The FSP problem:
  - A set J of n independent jobs needs to be scheduled on a set M of m independent machines
  - Job j requires p_{ij} units of time to be completed on machine i
  - Several constraints must be considered: the execution of a job cannot be interrupted, each machine can execute at most one job at a time, the order in which jobs are executed is the same

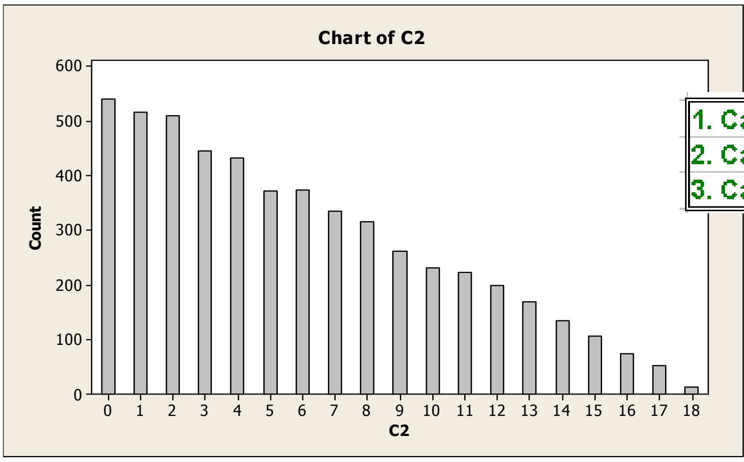


- Goal: find optimal permutation of jobs given a certain criterion (makespan for PFSP)
- Different approaches for FSP: optimization methods (small-size), heuristics (NEH) and meta-heuristics (GAs, TS, SA, GRASP, …)

Notice that stochastic times could also be considered! (e.g. Siemens)

8. Example 2: The FSP (2/4)

- NEH → jobs are ordered in decreasing order according to their total completion time on all the machines
- SS-GNEH introduces randomness in this process by using a triangular statistical distribution → jobs that take longer to complete will be more likely to be selected first, but all jobs in the list are potentially eligible.
- Notice: Each time SS-GNEH is run, a random feasible solution is obtained. By construction, chances are that this solution outperforms the NEH one → hundreds of 'good' solutions can be obtained after some seconds/minutes

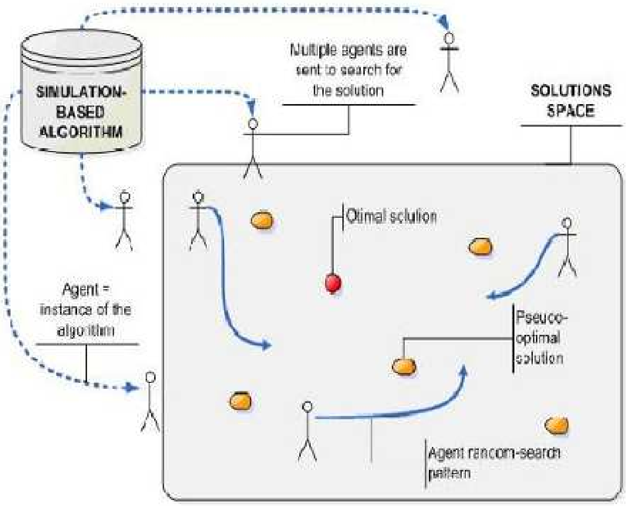


1. Calculate u ~ U(0,1)
2. Calculate X = b * (1 - Raiz(1-u))
3. Calculate pos = floor(X)

Pseudo-random number generation must be computationally efficient!

8. Example 2: The FSP (3/4)

- The main steps for SS-GNEH:
  1. Generate Randomized NEH solutions until you find one (our base) that outperforms the original NEH solution
  2. Keep applying a local search to the base solution from step 1 as long as you get improvements
  3. Update the best solution found so far if necessary
  4. Restart the process if time permits (30 ms x #jobs x # machines in our implementation)
  5. Run each instance with several different seeds for the random generator



Multi-Agent Approach

- The local search process:
  - Pick at random and without repetition one job from the list
  - Move the job at the end and apply Taillard acceleration to find the best position for it with respect to makespan
  - Repeat the 2 steps above n times, where n is the number of jobs

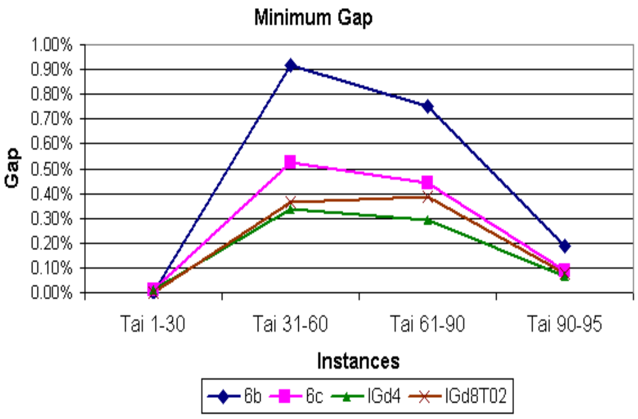
8. Example 2: The FSP (4/4)

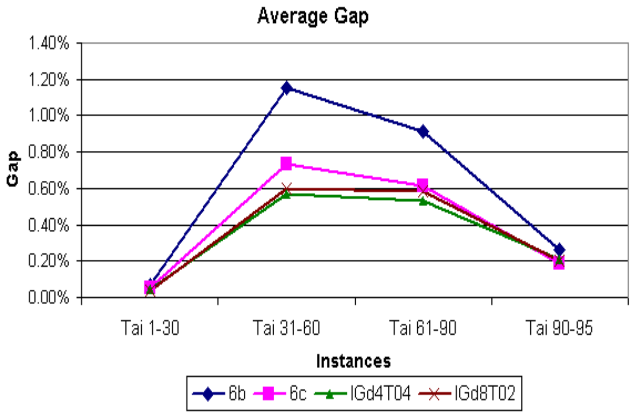
1. Test: 15 runs per instance with maxTime = 0.010s * nJobs * nMachines
2. Computer: Intel Xeon 2.0GHz 4GB RAM
3. Note: All algorithms have been implemented in Java (non-optimized code)

Summary of results:

	Average Gap					Minimum Gap			
	6b	6c	IGd4T04	IGd8T02		6b	6c	IGd4	IGd8T02
Tai 1-30	0.07%	0.05%	0.05%	0.04%		0.00%	0.01%	0.01%	0.00%
Tai 31-60	1.15%	0.74%	0.57%	0.60%		0.92%	0.52%	0.34%	0.37%
Tai 61-90	0.91%	0.62%	0.53%	0.59%		0.75%	0.44%	0.29%	0.39%
Tai 90-95	0.26%	0.18%	0.21%	0.20%		0.19%	0.09%	0.07%	0.08%

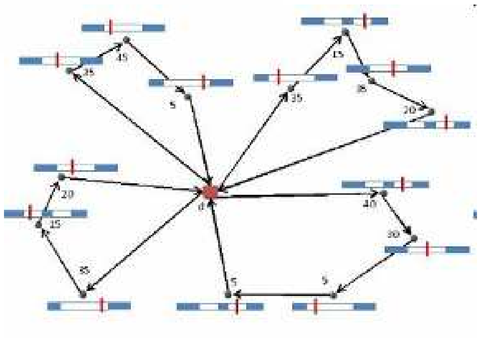
Juan, A.; Ruiz, R.; Mateo, M.; Lourenço, H.; Ionescu, D. (2010): "A Simulation-based Approach for Solving the Flowshop Problem". In *Proceedings of the 2010 Winter Simulation Conference*.

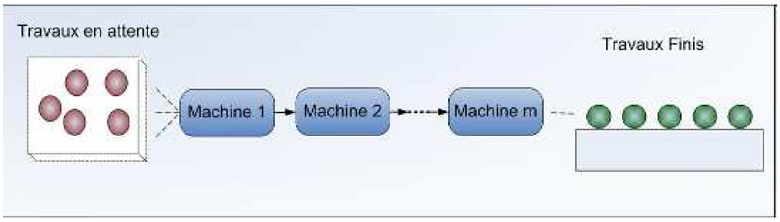




9. Future Work

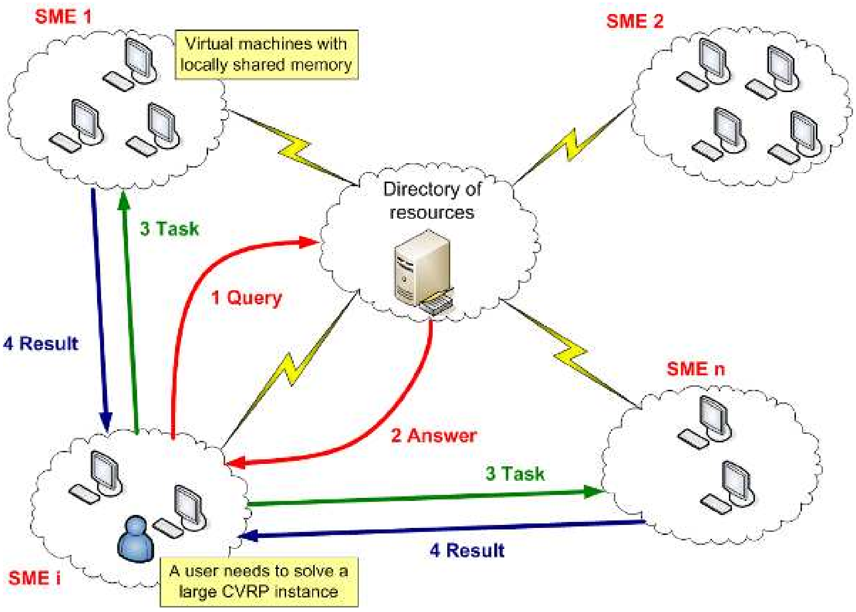
- Tesla GPUs (CUDA) → about 12,000 threads (for real-time solutions & large-size CVRPs)
- VRPTW, VRPSD, …
- Stochastic routing and scheduling problems
- Splitting with AI techniques
- Hybridization with Constraint Programming techniques, Lagrange Relaxation, etc.
- Parallel & Distributed computing (multi-agent approach)
- …





10. The Role of Distributed Computing

- Usually, small- and medium- enterprises (SMEs) in the logistics business lack technical expertise and high-tech computational resources.
- In such scenarios, two alternative DPCS approaches are possible: a) to use third-party resources on demand, i.e. a cloud system, or b) to employ idle computing capabilities of SME's desktop computers.
- Thus, it makes sense to spare resources from each computer and aggregate those resources into a computational environment where hundreds or even thousands of instances of a simple algorithm like the one presented here can be run simultaneously.



11. Conclusions

- We have discussed the use of probabilistic or stochastic algorithms for solving non-smooth combinatorial optimization problems.
- We propose the use of probability distributions, such as the Geometric or the Triangular ones, to add a biased random behavior to classical heuristics such as the Clarke and Wright Savings heuristic for the Vehicle Routing Problem or the NEH heuristic for the Flow Shop Scheduling Problem.
- By randomizing these heuristics, a large set of alternative good solutions can be quickly obtained in a natural and easy way.
- In some sense, this approach could be considered as a "Biased GRASP" (as far as we know, most existing GRASP only use uniform distributions).
- Some specific examples of this technique (VRP and PFSP) have been analyzed to illustrate the main ideas behind this approach.
- Future work relates to the use of parallel and distributed computing.

Conference on
Numerical Optimization and Applications in Engineering

13 – 15 October 2010

Centre de Recerca Matemàtica · Bellaterra, Barcelona

Using Multi-Start Randomized Heuristics to solve Non-Smooth and Non-Convex Optimization Problems

Thank You!

www.uoc.edu

Massachusetts Institute of Technology

A. Juan, D. Ionescu, J. Faulin, A. Ferrer

ajuanp@gmail.com | http://ajuanp.wordpress.com

Computer Science Department

Open University of Catalonia, SPAIN